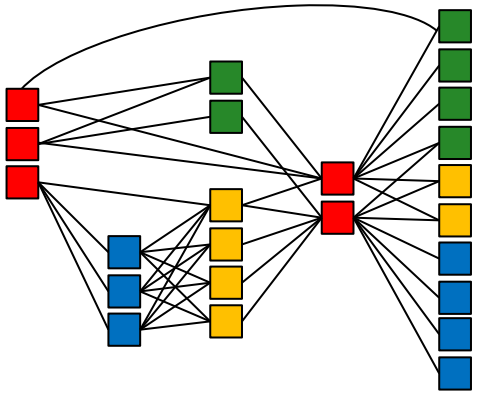
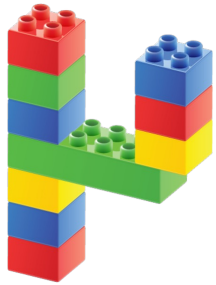
Automn School Series in Acoustics

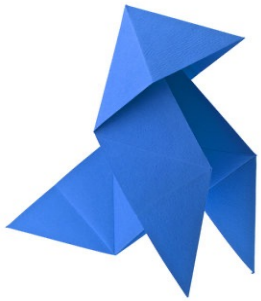
Lecture 1: Fundamentals of Machine Learning

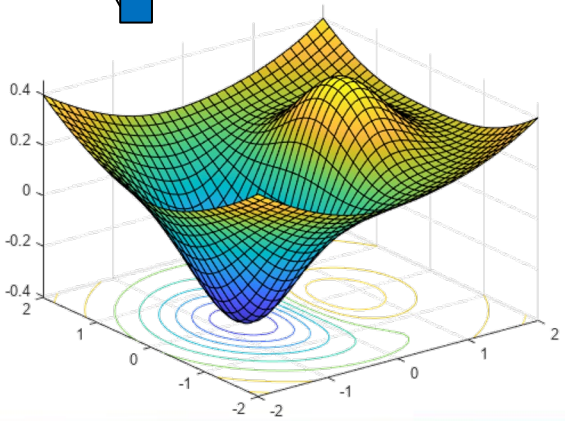
Eindhoven University of Technology
3-7 November 2025

Antoine Deleforge & Mathieu Gaborit









ASSA
AUTUMN SCHOOL SERIES IN ACOUSTICS

Outline

1. Introduction

   AI? Machine Learning? Neural Networks?

2. Machine Learning

   Definition, Characteristics, Types, Algorithms

3. Deep Neural Networks

   Definition, Motivation, How to Train Them

4. Limits of Machine Learning

   Overfitting, Data Biases

Then: Tutorial to put this in practice!

Artificial Intelligence

Machine Learning

Generative Models

Neural Networks

Deep Learning

ChatGPT, Grok, MidJourney,
Stable Diffusion, …

Artificial Intelligence

- What *is* intelligence? (Psychology / Philosophy)
- Used for communication / marketing / fundraising
- AI roots: collective imagination (Science fiction)
- AI ethics and safety (law, philosophy, politics)

"Applied" A.I.

- Solving « difficult » numerical problems *(optimization, game theory, …)*
- Representing knowledge and reasoning (logic, planning, inference, …)
- Managing uncertainty (representation, probabilities, decision procedures, …)
- Laguage and communication (natural language processing, text ↔ speech, avatars, robotics,…)

Symbolic A.I.

- Explicit knowledge representation
- Rule-based methods
- *Deep Blue* chess engine (1997)
- High-level programming languages (Python, mathematica)

Machine Learning

- **Algorithms** that (implicitly) build **models** from **data** to achieve **tasks**

Generative Models

- The model is a **probability distribution** that can be **sampled**

Neural Networks

- Class of **parameterized** families of **non-linear** functions

PINNs …

Deep Learning

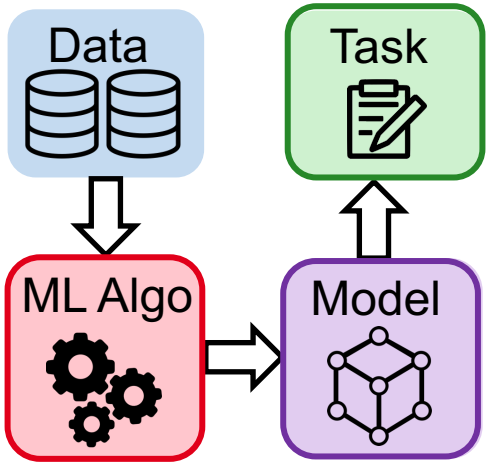
- « Deep » neural networks applied to machine learning

"A.I." / AI model (since ~2023)

ChatGPT, Grok, MidJourney, Stable Diffusion, …

Definition

The study of **Algorithms** that build **Models** from **Data** to achieve **Tasks**.



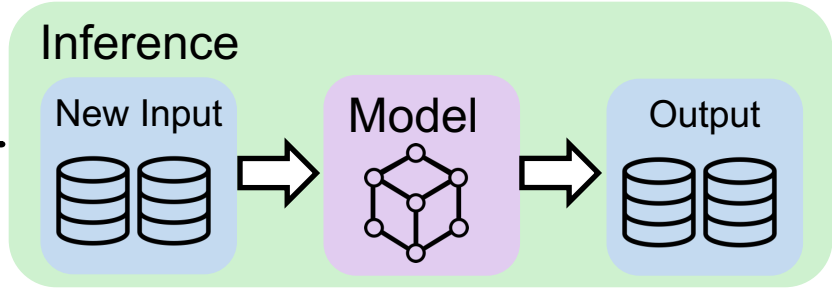
Model (in machine learning):

- A *"program created by a program"*
- Can be **deterministic** or **stochastic**
- **Data-Driven** as opposed to **Physics- / Knowledge-Driven** model

Not considered ML, even if the same Task is performed!

Tasks:

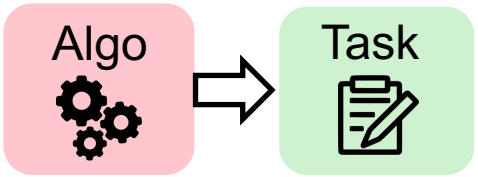
- Sort / Visualize / Represent the **Data**
- **Infer** from **new input Data:**
  - Estimate explanatory quantities
  - Generate, complete, predict
  - Transform, convert, translate
  - Decide

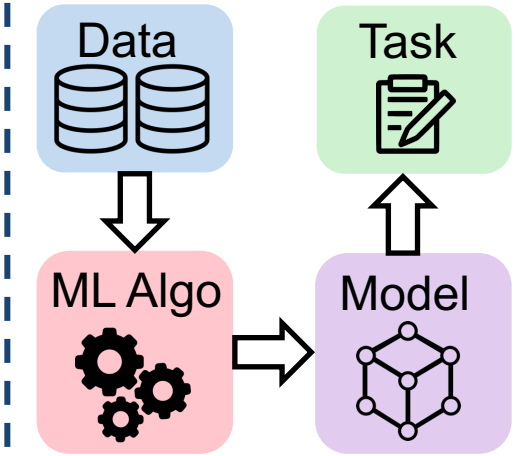


Contrast it with *symbolic* A.I.

```
def add_numbers(a, b):
    return a + b
```

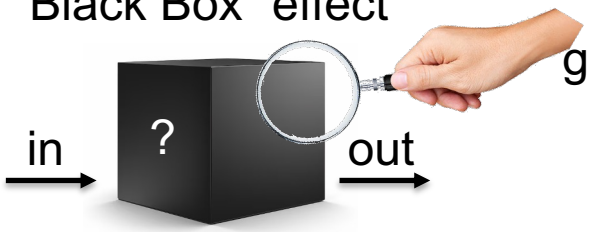
```
def square(num):
    result = num * num
    return result
```





- ML models do not necessarily contain **explicit** concepts with **meanings**

"Black Box" effect



Explainability / Interpretability / Theoretical guarantees for models = a rich subfield of ML
=> but done ***a posteriori***

- Is Machine Learning ***enough*** for A.I.?
  - A heated debate! Ex: Yann LeCun vs. Gary Marcus (NYU)
  - **Neuro-symbolic A.I.** aims at combining Symbolic A.I. and Neural Networks
  - Attractive approach: Hybridizing **physics-** and **data-**driven models

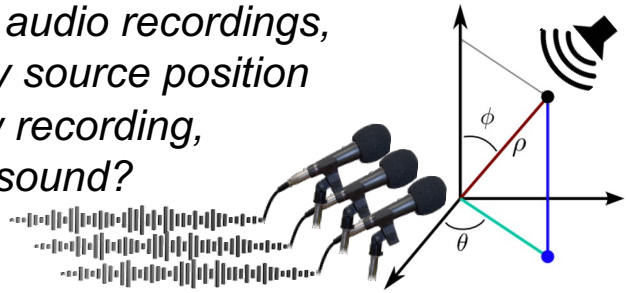
Types of ML based on available data

- Supervised Learning

Ex 1: Sound Source Localization

- A dataset of audio recordings, annotated by source position

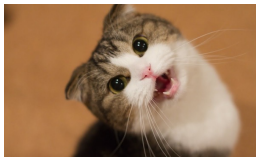
- Given a new recording, where's the sound?

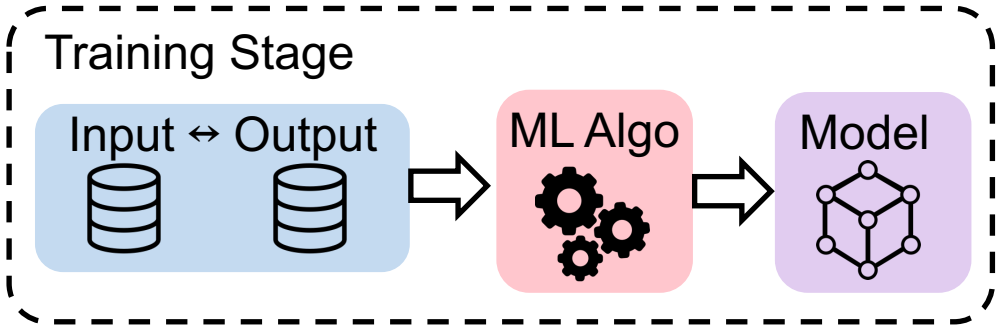


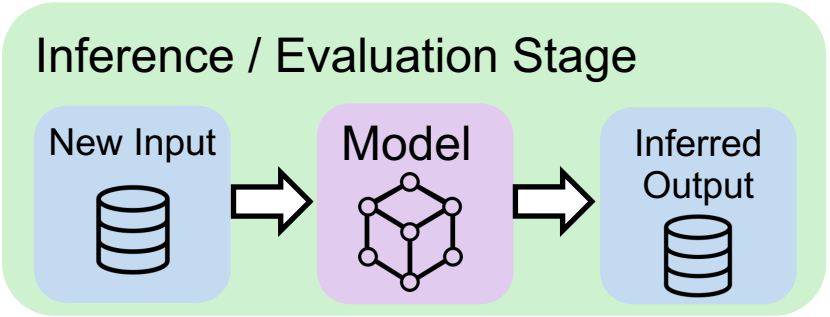
Ex 2: Sound Event Recognition

- An annotated dataset of cat sounds and baby sounds

- Given a new recording, is it a cat or a baby?







Types of ML based on available data

- Unsupervised Learning

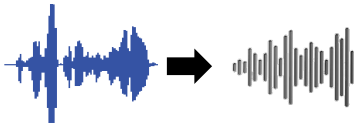
Ex 1: Speaker diarization

- Given a 1h audio recording of a meeting between 3 unknown people, annotate the speaking turns

Ex 2: Anomaly detection

- Identify a gas leak, or a train rail defect from acoustic recordings (indutry monitoring)

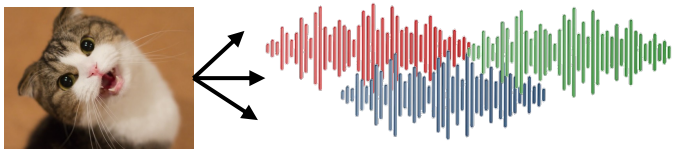
Ex 3: Audio compression

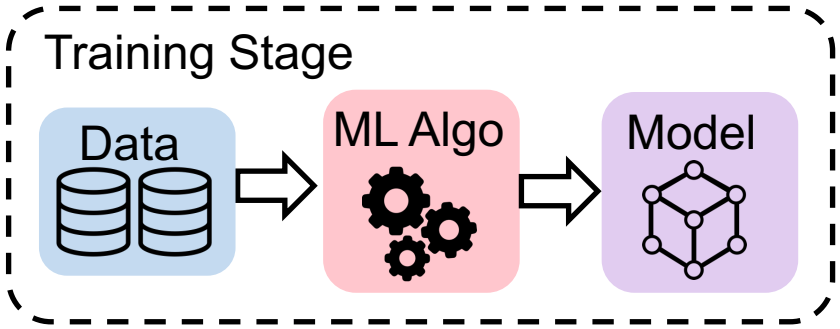


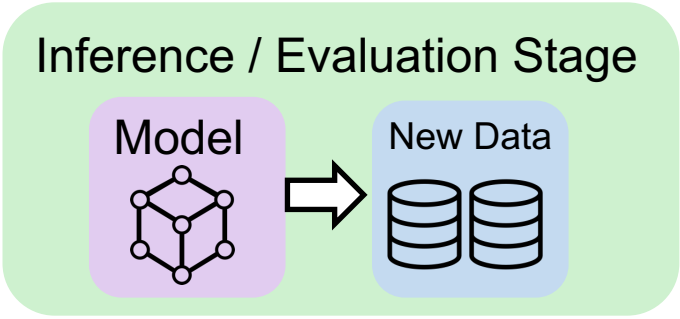
- Obtain a low bitrate representation of music tracks while preserving perceptual fidelity

Ex 4: Audio generation

- Given a dataset of cat recordings, generate new plausible cat sounds







OUTPUT ↓

Supervised Learning

Labeled training data $\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$ $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$ Input ↔ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N]$ Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → Estimate $\tilde{\boldsymbol{y}}$

Unsupervised (representation) Learning

Unlabeled training data $\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$ $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$ Input

Training

Learned Model f_θ → Representation $[\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N]$

CONTINOUS

Regression

$\tilde{\boldsymbol{x}}$ → $\tilde{\boldsymbol{y}} = (\theta, \phi)$

Dimensionality Reduction

$\{\boldsymbol{x}_n\}_{n=1}^{N} \rightarrow \{\boldsymbol{z}_n\}_{n=1}^{N}$

DISCRETE

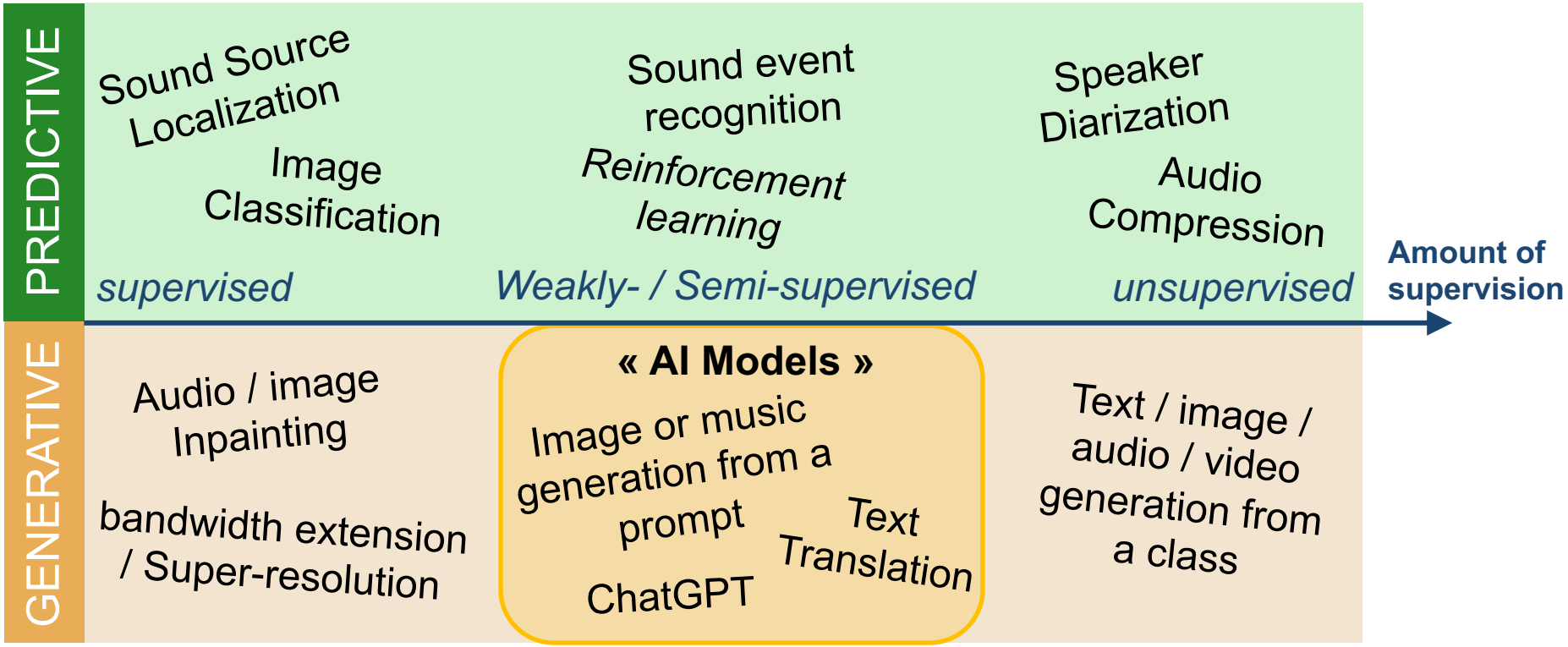
Classification

$\tilde{\boldsymbol{x}} \rightarrow \tilde{\boldsymbol{y}} \in \left\{ \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix} \right\}$

Clustering

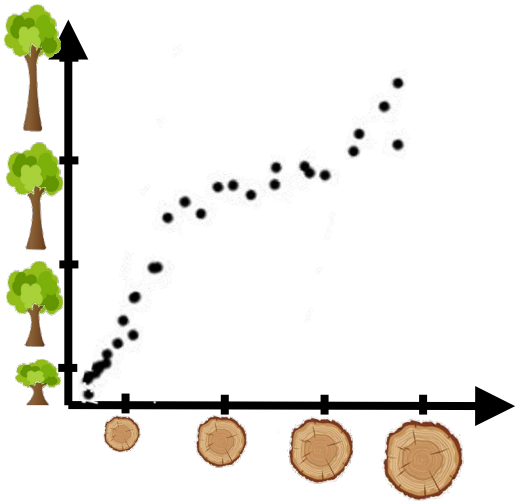
$\{\boldsymbol{x}_n\}_{n=1}^{N} \rightarrow \{\boldsymbol{z}_n\}_{n=1}^{N} \subset \left\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix} \right\}$

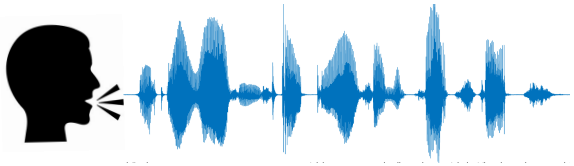
A “Continuum” of Supervision

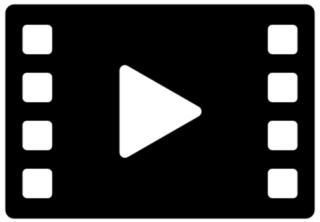


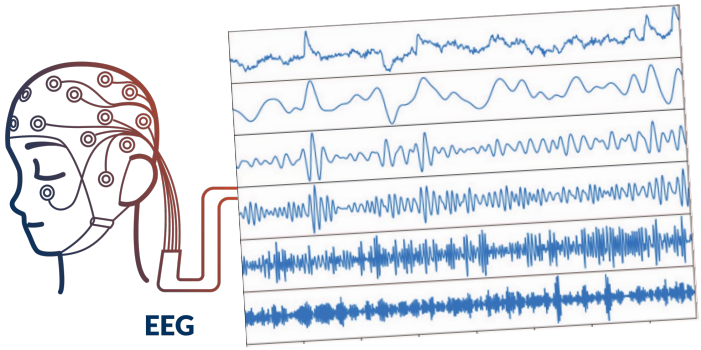
Types of data used in machine learning

- Numerical Data / Continuous Data / Signals / Vectors in $\mathbb{R}^D$









Types of data used in machine learning

- Categorical Data

ImageNet Labels
- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck
- …

```
Sex ∈ {male, female, other}
ZIP ∈ {67000, 75009, …}
Country ∈ {France, Spain, …}
```

- Text Data

Artificial intelligence

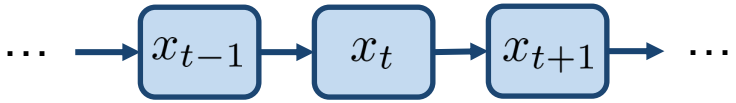
Article Talk

From Wikipedia, the free encyclopedia

Artificial intelligence (**AI**) is intelligence—perceiving, synthesizing, and inferring information—demonstrated by machines, as opposed to intelligence displayed by non-human animals and humans. Example tasks in which this is done include speech recognition, computer vision, translation between (natural) languages, as well as other mappings of inputs.

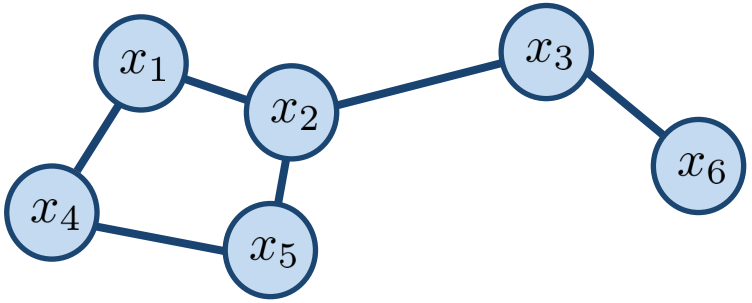
AI applications include advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or creative tools (ChatGPT and AI art), automated decision-making and competing at the highest level in strategic game systems (such as chess and Go).[1]

- Time series



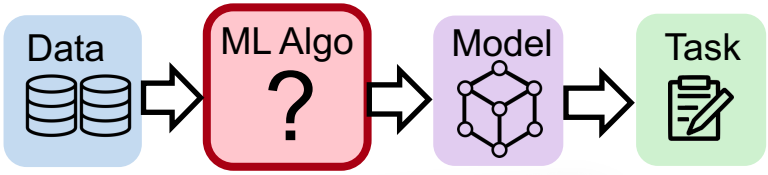
- Heterogenous / Tabular Data

- Data on graphs



Machine learning algorithms

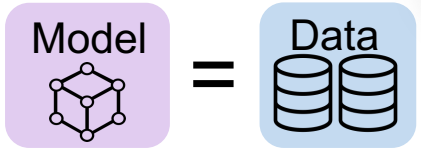
How to make a program that makes a program?



- Search in the set of all python functions?
  - 🚫 a gigantic combinatorial set

```
def add_numbers(a, b):
    return a + b
```

```
def square(num):
    result = num * num
    return result
```

- Simply **memorize** the data 

?
  - *"Lazy Learning"*
  - Ex: k-NN, look-up table, naive Bayes, case-based
  - Slow, large storage, non-robust

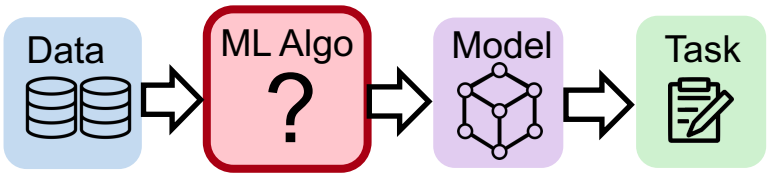
- **Model Fitting:**
  Find the *best* model within a **parameterized family** $\mathcal{F} = \{m_\theta\}_{\theta \in \Theta}$

Dataset

- $m_\theta =$ a jean
- $\theta =$ its (width, length)
- $\mathcal{F} =$ the shelves

Machine learning algorithms

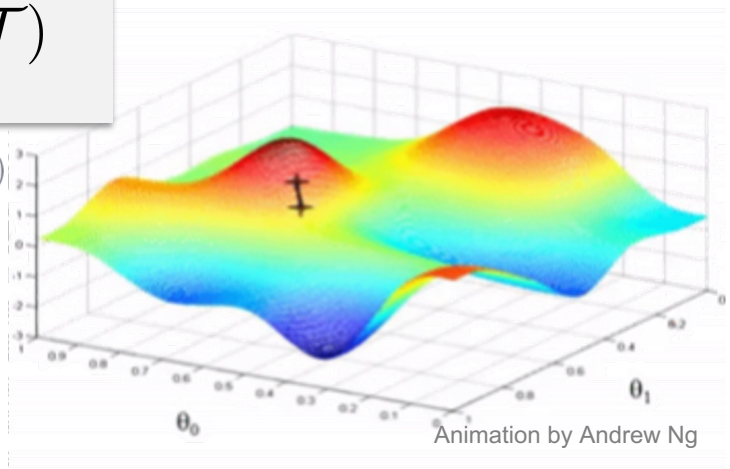
How to make a program that makes a program?



- The model is found by minimizing a ***total loss / cost function*** L over the set of parameters, for a given training **dataset** $\mathcal{T}$:

$$\hat{m} = m_{\hat{\theta}} \quad \text{where} \quad \hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}}\, L(m_\theta, \mathcal{T})$$

- Examples:
  - *mean localization error in degrees*
  - *mean classification error*
  - *mean signal-to-error ratio in dB*
  - *negative likelihood of observed data*

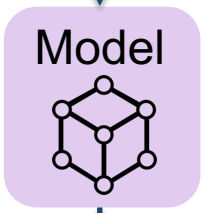


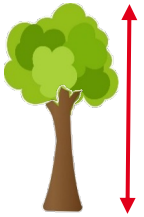
- The total loss L is designed based on the **task**, the **data**, and the chosen family of **models**. It measures the **fit** of $m_{\hat{\theta}}$ for these data and task.
- Most modern machine learning **algorithms** can be interpreted as minimizing a loss, using **optimization**.

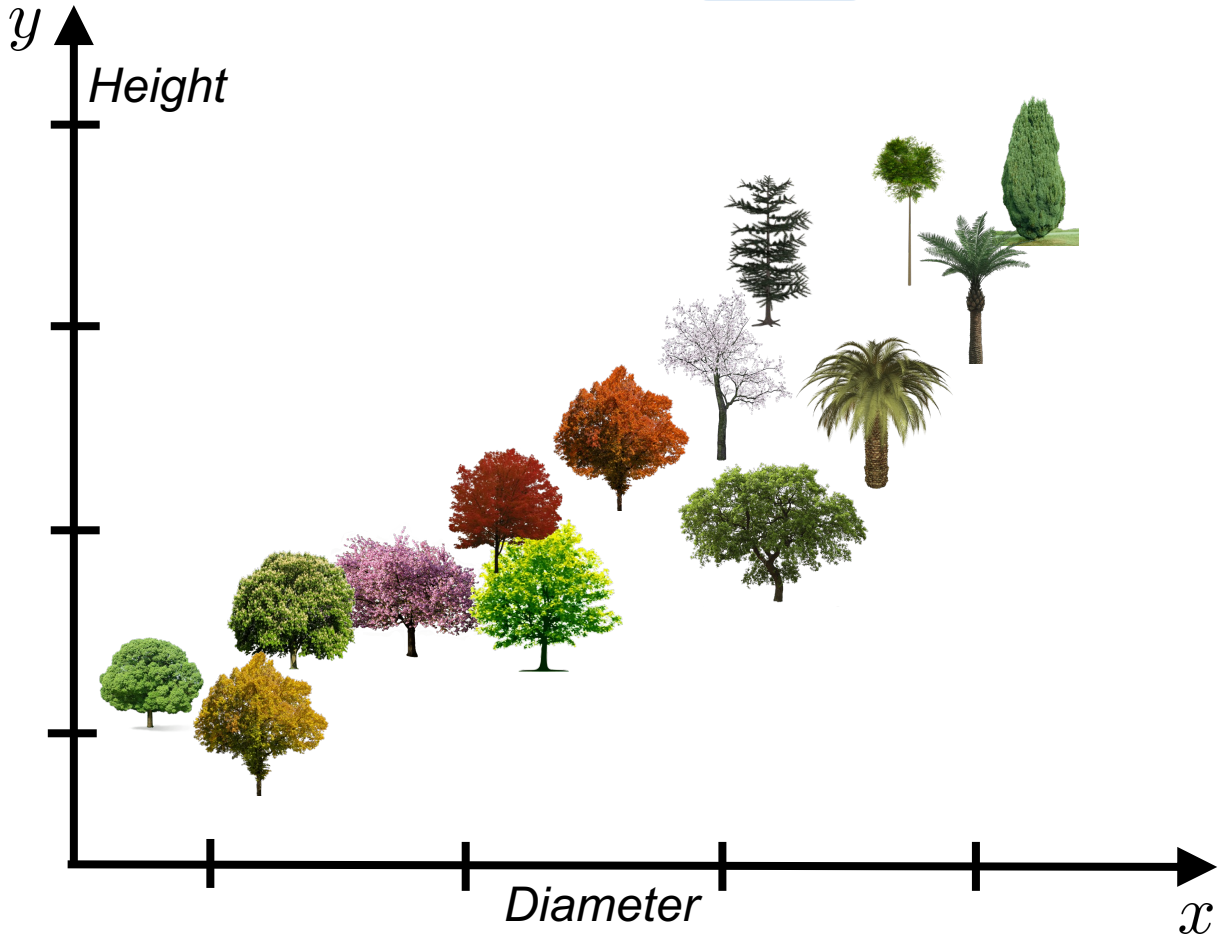
What is a model?

Example:

Tree diameter 

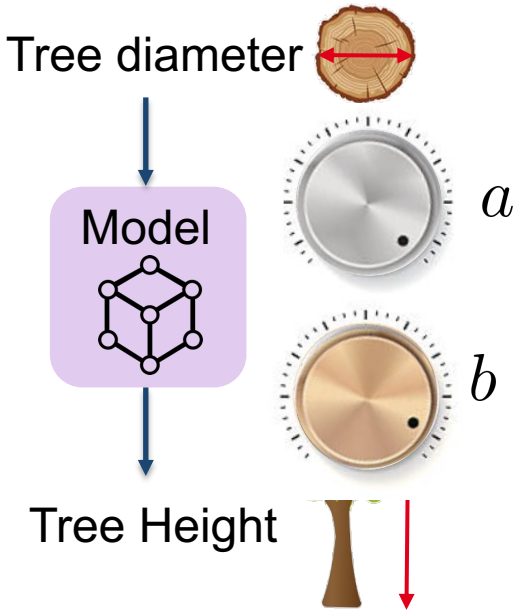


Tree Height 



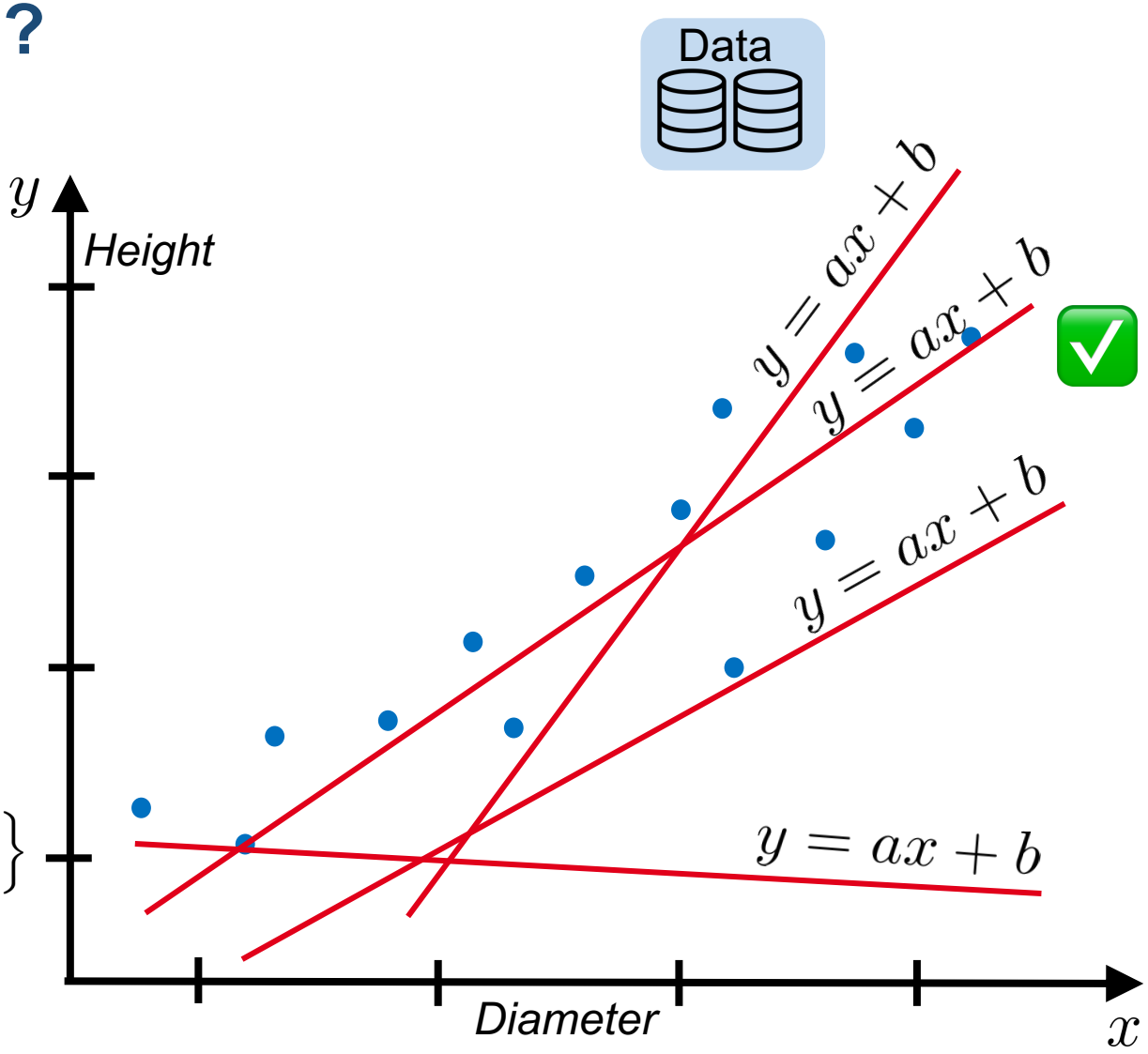
What is a model?

Example:



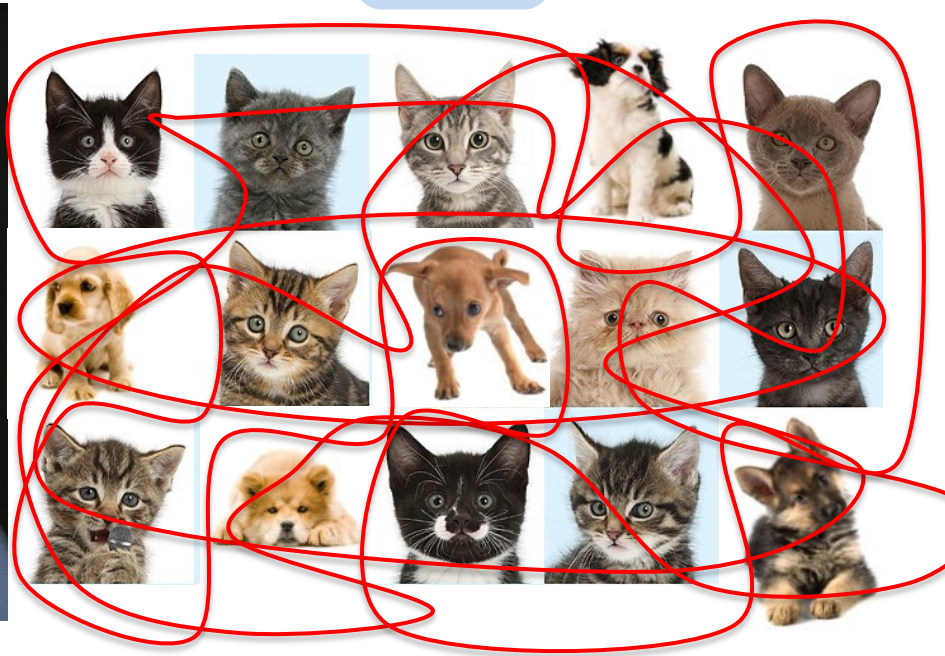
$$\underset{a,b}{\operatorname{argmin}} \left\{ \operatorname{mean}_i (ax_i + b - y_i)^2 \right\}$$

« The line is as close as possible to the points »



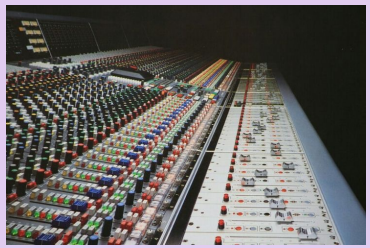
What is a model?

Training stage



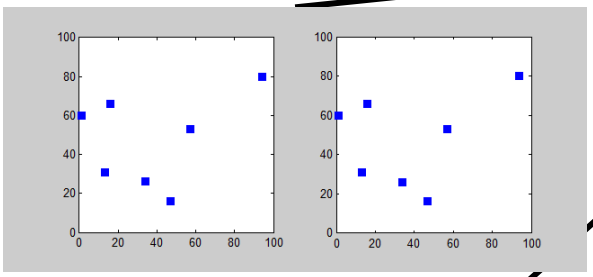
Inference / evaluation stage

Frozen model

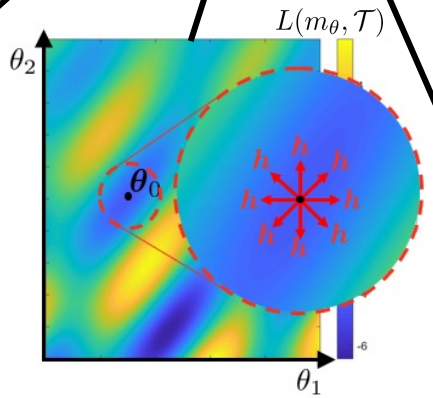


It's a cat !

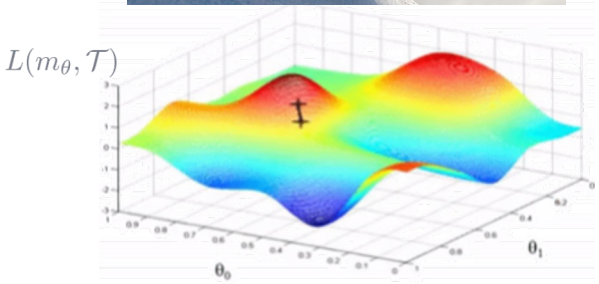
Optimization Techniques



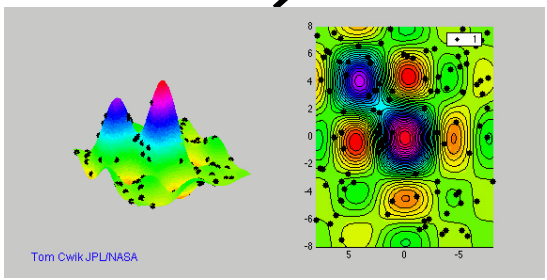
1. Brute Force



3. Zeroing gradient



5. Gradient Descent



2. Population-Based

$$\theta_1^{(i+1)} = \underset{\theta_1}{\operatorname{argmin}}\ g(\theta_1, \theta_2^{(i)}, \ldots, \theta_P^{(i)})$$
$$\theta_2^{(i+1)} = \underset{\theta_2}{\operatorname{argmin}}\ g(\theta_1^{(i+1)}, \theta_2, \ldots, \theta_P^{(i)})$$
$$\vdots \qquad\qquad \vdots$$
$$\theta_P^{(i+1)} = \underset{\theta_P}{\operatorname{argmin}}\ g(\theta_1^{(i+1)}, \theta_2^{(i+1)}, \ldots, \theta_P)$$

4. Alternate Minimization

Artificial Intelligence

“Applied” A.I.

Symbolic A.I.

Machine Learning

- **Algorithms** that (implicitly) build **models** from **data** to achieve **tasks**

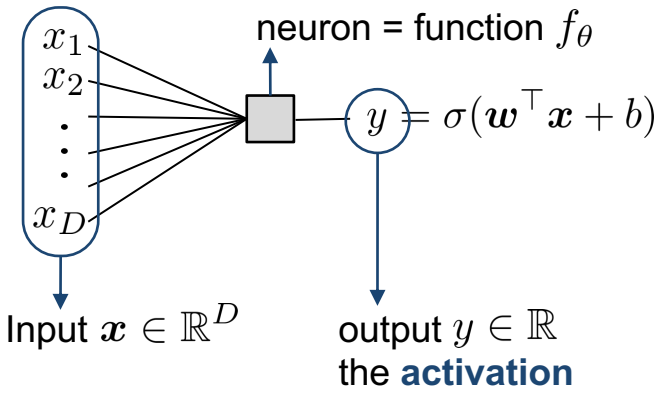
- Different **flavors** depending on available data and task (sup./unsup.)
- Main approch = **Model Fitting**: find a model that **minimizes a loss** within a **parameterized** family by **optimization**

Neural Networks

Deep Learning

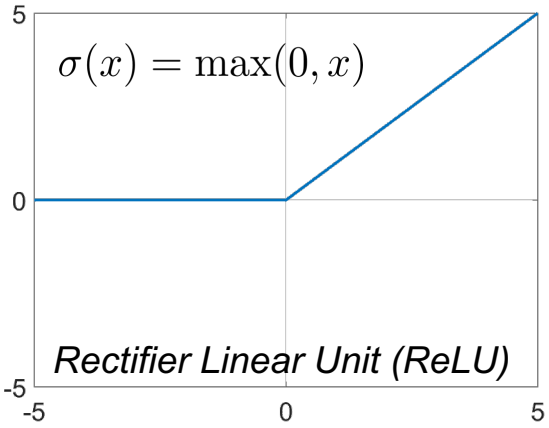
Artificial Neural Networks

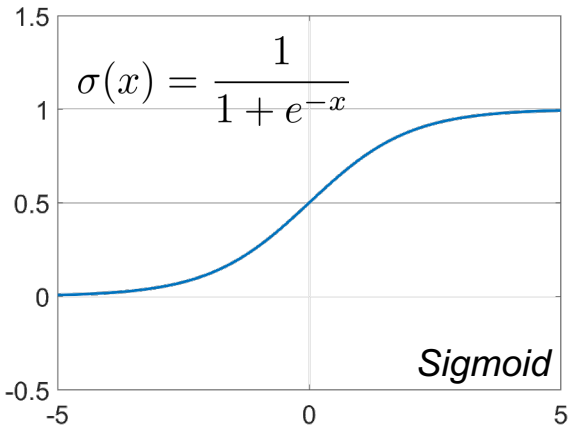
- **Artificial Neuron** = a multiple-input, single-output, **parametric, nonlinear function**



- $\boldsymbol{w}^\top \boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{d=1}^{D} w_d x_d$, the ***dot product***
- $\boldsymbol{w} = [w_1, \ldots, w_D]^\top \in \mathbb{R}^D$, the ***weights***
- $b \in \mathbb{R}$, the ***bias***
- $\boldsymbol{w}^\top \boldsymbol{x} + b$: the ***pre-activation***
- $\theta = (\boldsymbol{w}, b) \in \mathbb{R}^D \times \mathbb{R}$, the ***parameters***
- $\sigma : \mathbb{R} \to \mathbb{R}$, the ***activation function*** or ***non-linearity***.

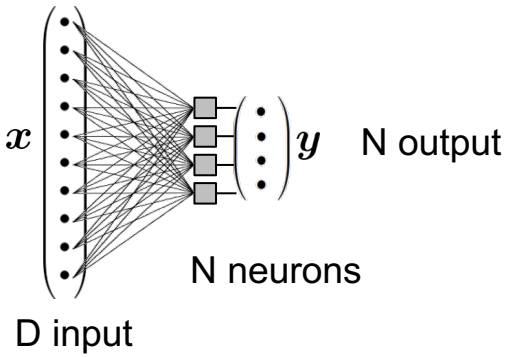
Examples of activation functions:





Artificial Neural Networks

- **Simple perceptron**: multiple neurons attending the same input



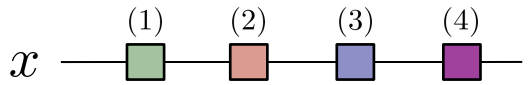
- We have: $y_n = \sigma(\boldsymbol{w}_n^\top \boldsymbol{x} + b_n)$
- Or in matrix form: $\boldsymbol{y} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$, where

$$\mathbf{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \boldsymbol{w}_2^\top \\ \vdots \\ \boldsymbol{w}_N^\top \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,D} \\ w_{2,1} & w_{2,2} & \dots & w_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N,1} & w_{N,2} & \dots & w_{N,D} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}, \quad \theta = (\mathbf{W}, \boldsymbol{b}) \in \mathbb{R}^{N \times D} \times \mathbb{R}^N$$

- Note: In this notation, σ is applied **elementwise** to a vector

Artificial Neural Networks

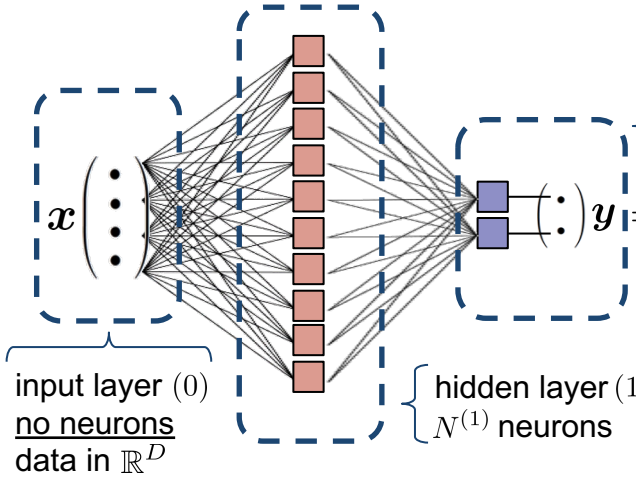
- Neurons can be **chained** together



$$y = f_\theta(x) = \sigma\left(b^{(4)} + w^{(4)} \cdot \sigma\left(b^{(3)} + w^{(3)} \cdot \sigma\left(b^{(2)} + w^{(2)} \cdot \sigma\left(b^{(1)} + w^{(1)} \cdot x\right)\right)\right)\right)$$

$$\theta = \{b^{(1)}, w^{(1)}, b^{(2)}, w^{(2)}, b^{(3)}, w^{(3)}, b^{(4)}, w^{(4)}\} \in \mathbb{R}^8$$

- **Simple Perceptron** (single hidden layer)



$$= f_\theta(\boldsymbol{x}) = \sigma\left(\boldsymbol{b}^{(2)} + \mathbf{W}^{(2)} \cdot \sigma\left(\boldsymbol{b}^{(1)} + \mathbf{W}^{(1)} \cdot \boldsymbol{x}\right)\right)$$

$$\theta = \{\boldsymbol{b}^{(1)}, \mathbf{W}^{(1)}, \boldsymbol{b}^{(2)}, \mathbf{W}^{(2)}\} \in \mathbb{R}^{N^{(1)}} \times \mathbb{R}^{N^{(1)} \times D} \times \mathbb{R}^{N^{(2)}} \times \mathbb{R}^{N^{(2)} \times N^{(1)}}$$

$$\#\theta = N^{(1)} + N^{(1)}D + N^{(2)} + N^{(1)}N^{(2)}$$

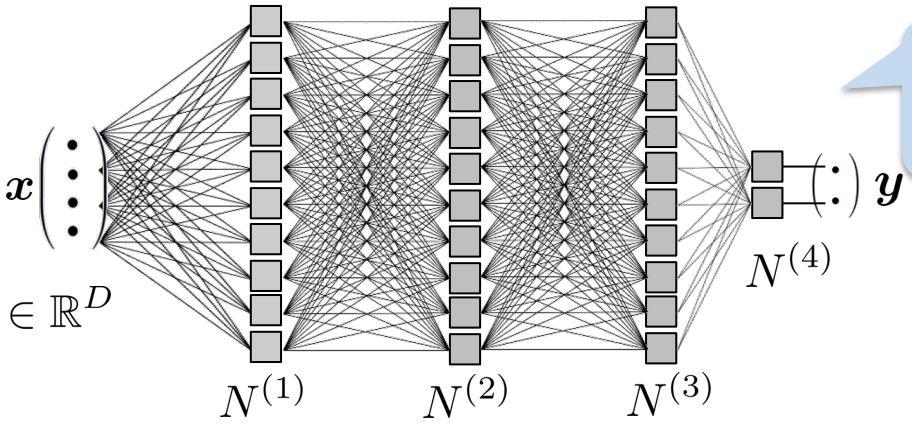
***Q:** what happens if we choose σ to be **linear** ?*
- *f_θ is then linear*

Universal Approximation Theorem [Hornik, Stinchcomb, White, 1991]: *"A **single hidden layer** neural network with any **"sigmoid-like"** activation function and with a **linear output** unit can approximate any continuous function arbitrarily well, for **sufficiently large** $N^{(1)}$."*

… But the required $N^{(1)}$ may be **exponential** in the input dimension D .

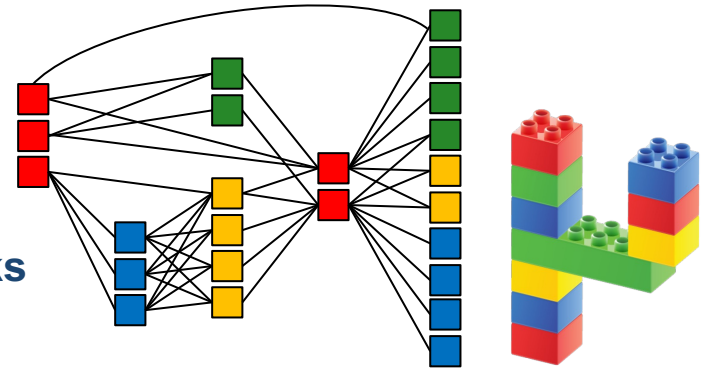
Artificial Neural Networks

- **Multilayer Perceptron** (**MLP**) with 3 hidden layer (***Depth 4***)



Q:
Suppose $D = N^{(1)} = N^{(2)} = N^{(3)} = N^{(4)} = N$, *how many parameters?*

$$\#\theta = 4(N + N^2) = \mathcal{O}(N^2)$$



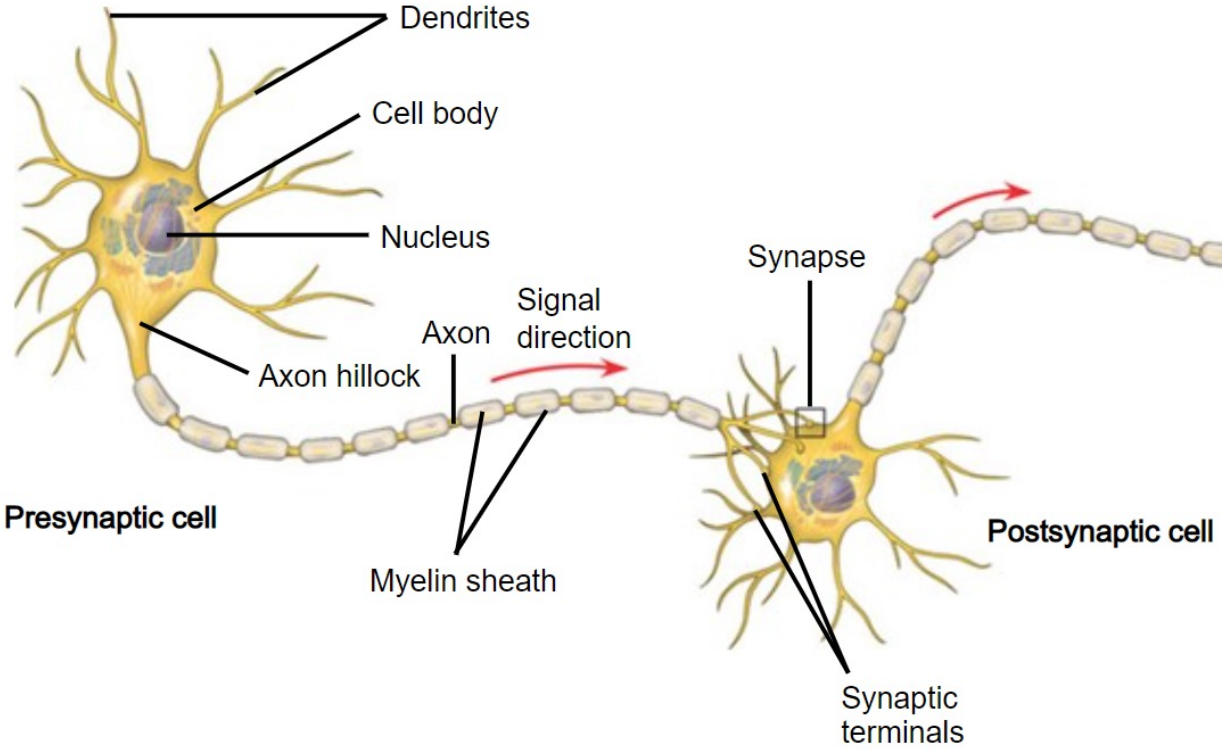
- Artificial neurons, as **elementary computing units**, can be combined in many different ways
- No cycle in the graph = **Feedforward Neural Networks**
- We call a given network of neurons an **architecture**
- Neural Networks form a very flexible **class** of **parameterized families** of **non-linear functions:** $\{\mathcal{F}_i\}_i$ where $\mathcal{F}_i = \{f^i_\theta\}_\theta$

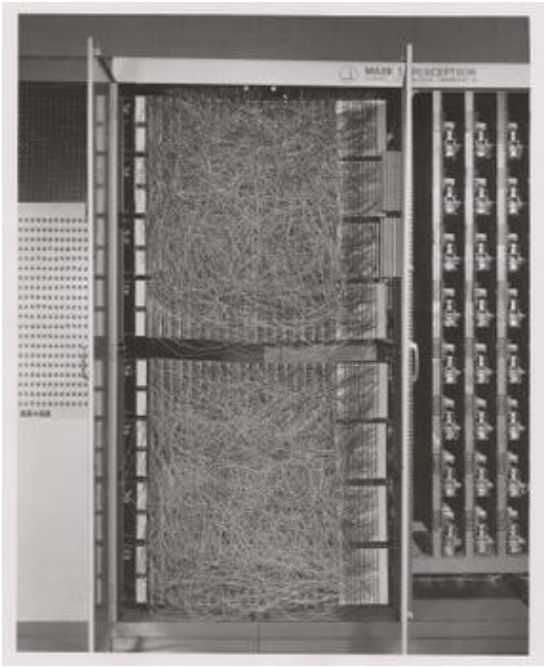
Artificial neural networks with 2 or more hidden layers are called **Deep Neural Networks** (DNNs)

Why Deep?

1) Biological inspiration

Frank Rosenblatt. « The perceptron: A probabilistic model for information storage and **organization in the brain** ». ***Psychological Review*** (1958).

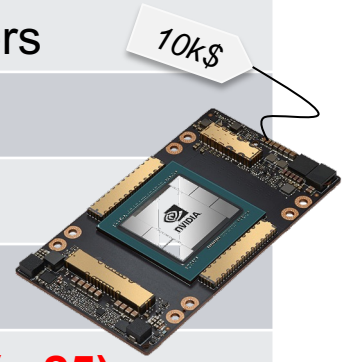




Why Deep?

1) Biological inspiration

Human Brain	Artificial Neural Network
~ 86 billion neurons	100k - 1 billion neurons
~ 7,000 synapse connections per neuron (~600 trillion connections)	3 - 1,000 connections per neuron 1 million - 1 trillion parameters
Massively parallel	(Mostly) sequential
Asynchronous	Synchronous
Very plastic architecture	(Mostly) fixed architecture
12 W of power	Nvidia A100 GPU : 300 W* **(x 25)**
Biological neuron = extremely complex** and not fully understood	Artificial Neuron = a weighted sum and a threshold.
3.7B year evolution + experiencing life	Designed by humans, trained on Internet



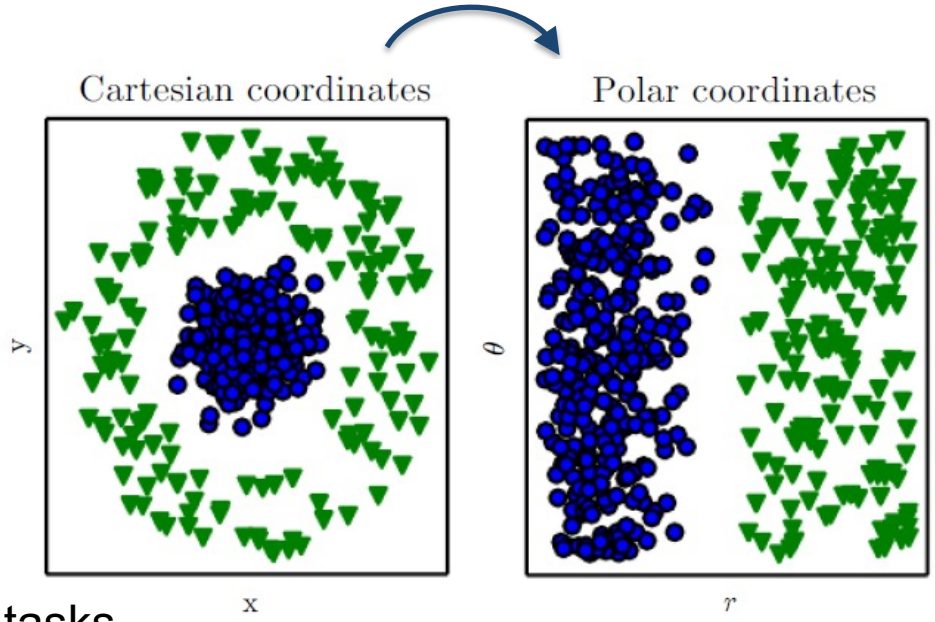
* - A chat session with chatGPT mobilizes roughly a full A100. Training GPT-3 is estimated to have taken ~95 years of A100.
- GPT-3 has ~135 billion parameters and roughly ~0.25 billion “neurons”.

** [Beniagev et al. 2021]: a single bio-neuron is well approximated by a 1000-neuron ANN of depth 5-8

Why Deep?

2) Bypassing feature engineering

- "Classical" machine learning relies heavily on **hand-crafted features**
- Such pre-processing allows successfully applying (combinations of) **simple / low-parametric** models
- But **difficult to do by hand** for many tasks

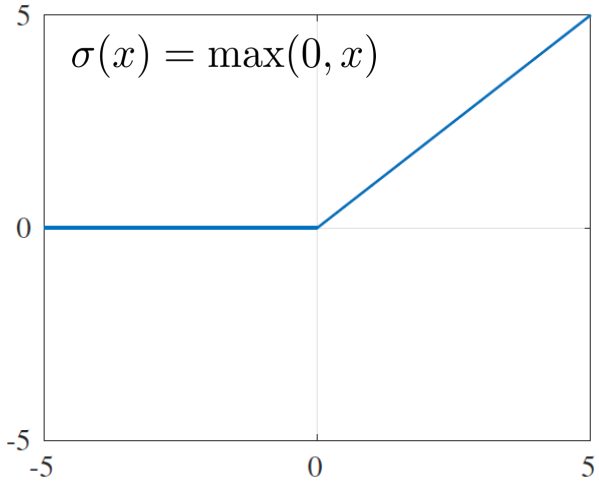


- **Sufficiently large DNNs** (and in particular deep **convolutional** nets) tend to **learn features** automatically. ➔ The ***deeper*** the ***higher*** level.
- Can be successfully applied to **raw signals** (e.g. image pixels, audio samples)

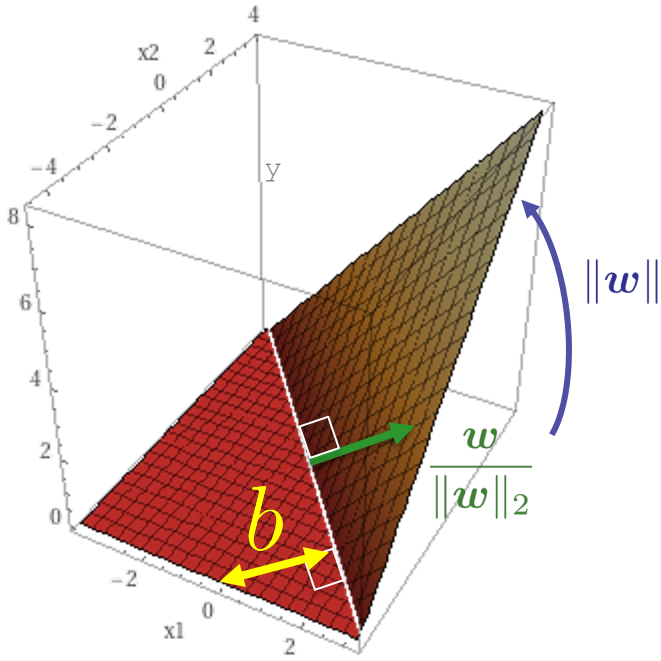
Why Deep?

3) The "Origami Effect"

- ReLU activation:



- For a 2D input:
$y = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$



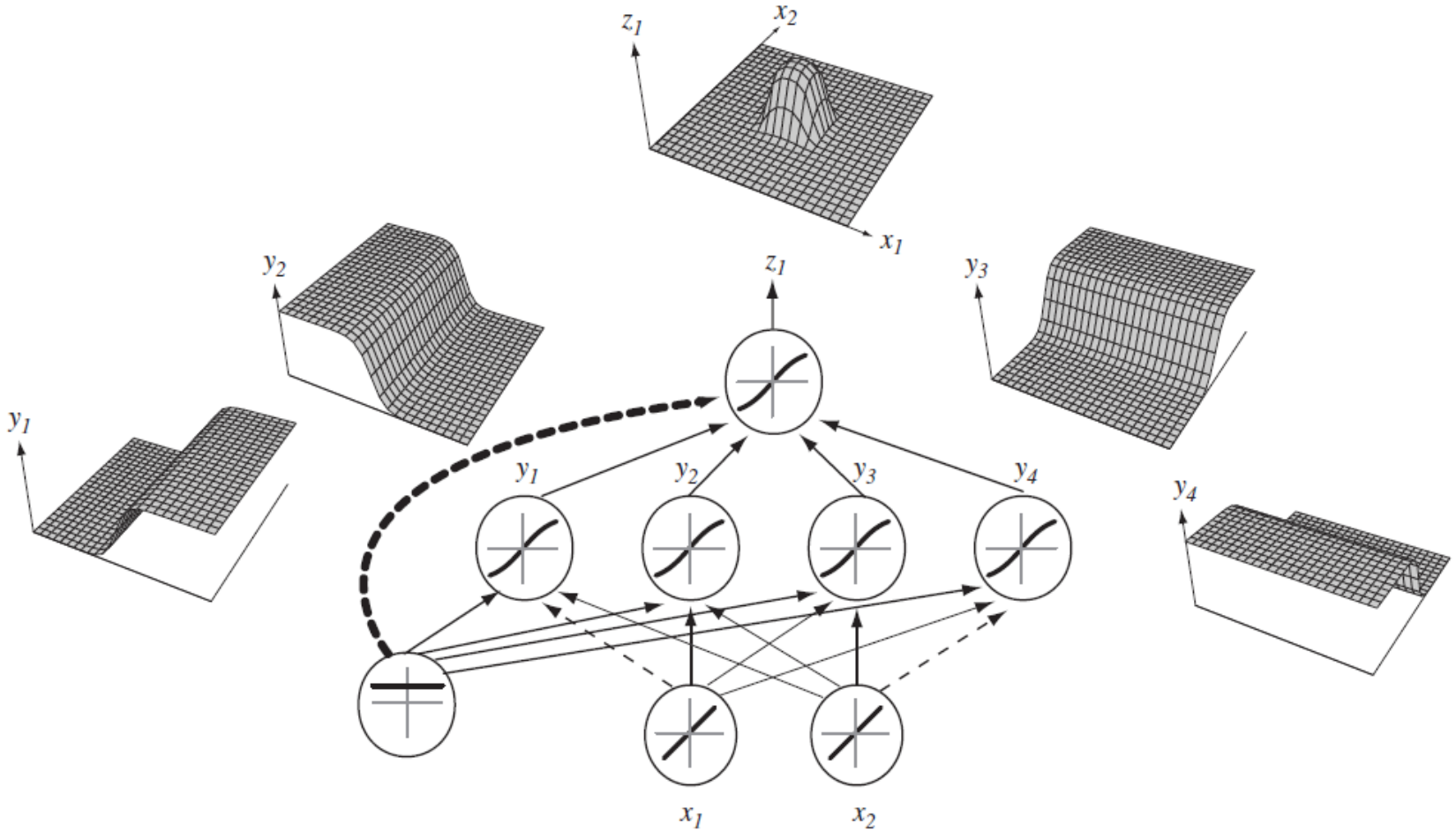
Theorem [G. Montufar et al., 2014]: *the max. number of linear regions modeled by a piecewise linear network (i.e., a network with ReLU neurons) with D inputs, L layers, and N units per layer is in the order of*

$$N^D \left(\frac{N}{D}\right)^{D(L-2)}$$

, i.e., the model capacity is ***exponential in the depth*** *L, i.e., in the* ***model size*** *(recall it is O(LN²))*

Why Deep?

3) The “Origami Effect”: Illustration with Sigmoids



How to train a DNN?

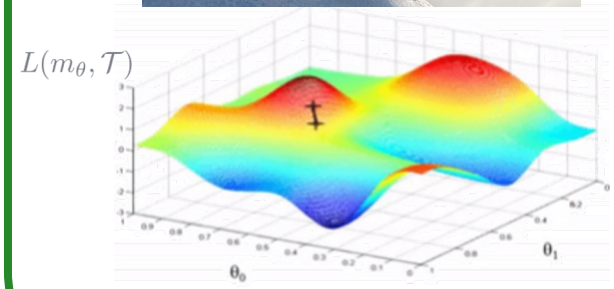
1. Brute Force

3. Zeroing gradient

2. Population-Based

$$\theta_1^{(i+1)} = \underset{\theta_1}{\operatorname{argmin}}\ g(\theta_1, \theta_2^{(i)}, \ldots, \theta_P^{(i)})$$
$$\theta_2^{(i+1)} = \underset{\theta_2}{\operatorname{argmin}}\ g(\theta_1^{(i+1)}, \theta_2, \ldots, \theta_P^{(i)})$$
$$\vdots \qquad\qquad \vdots$$
$$\theta_P^{(i+1)} = \underset{\theta_P}{\operatorname{argmin}}\ g(\theta_1^{(i+1)}, \theta_2^{(i+1)}, \ldots, \theta_P)$$

4. Alternate Minimization



5. Gradient Descent

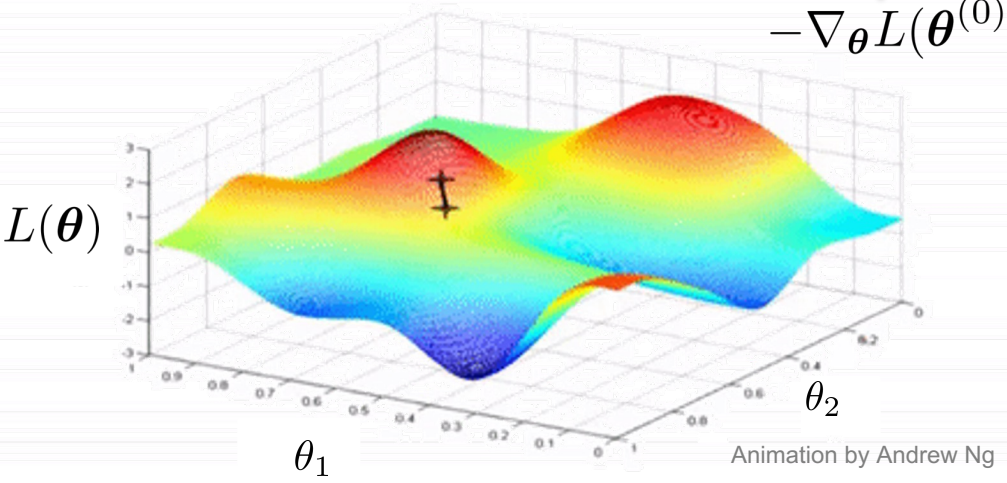
This one !

Gradient Descent

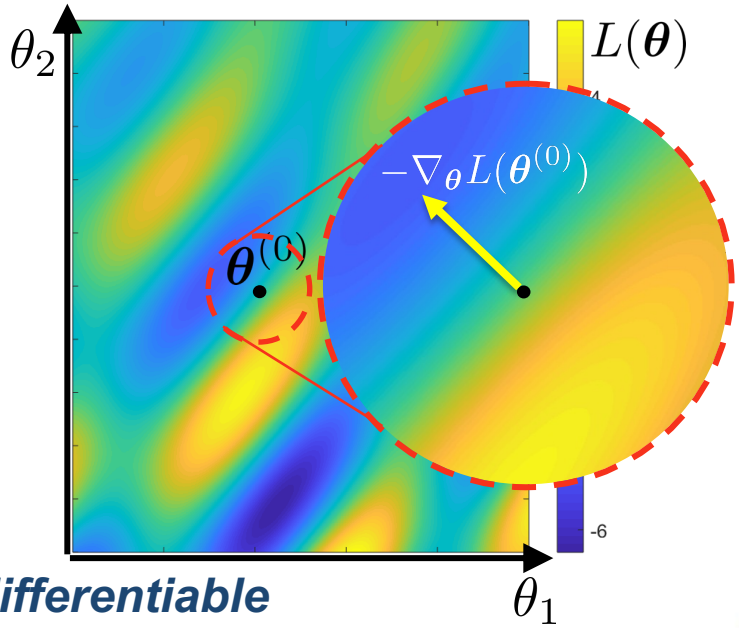
Intuition:

- Start from an initial **parameter vector** $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^P$
- From here, follow the ***direction of steepest descent***
- Stop when things look ***flat***

$$-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(0)}) = -\left[\frac{\partial L}{\partial \theta_1}\Big|_{\theta_1^{(0)}}, \cdots, \frac{\partial L}{\partial \theta_D}\Big|_{\theta_D^{(0)}}\right]^\top$$



Animation by Andrew Ng

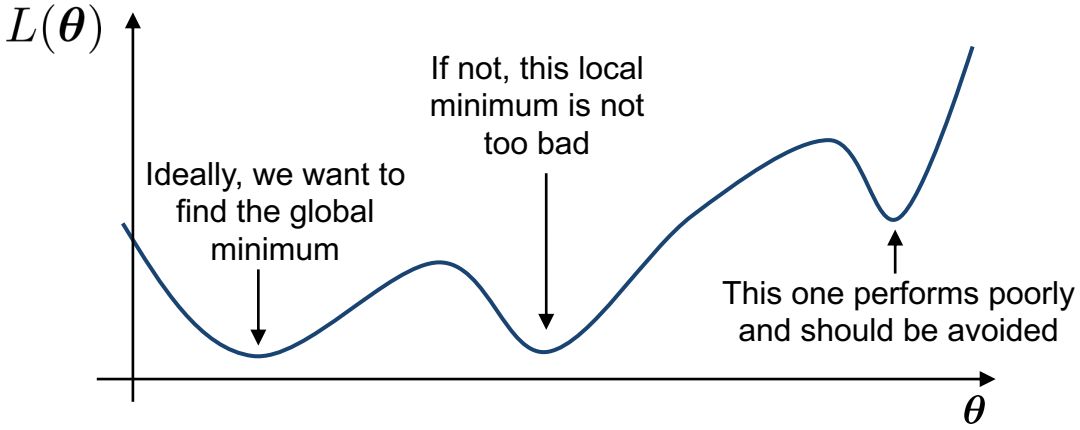


- The updates are: $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(i)})$
- Requires the function to be (almost everywhere) ***differentiable***

Gradient Descent

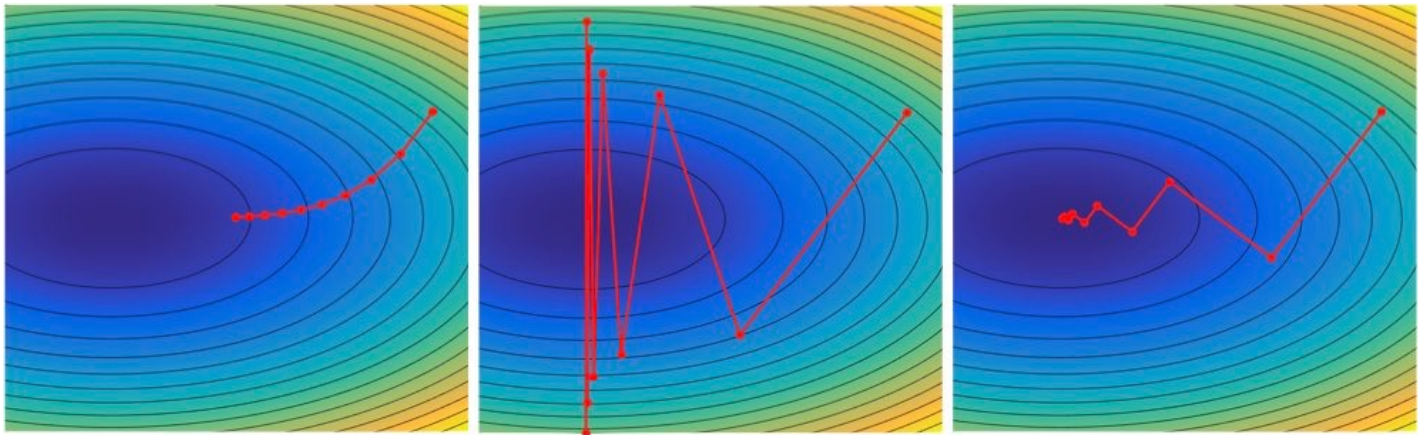
$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \epsilon \nabla_\theta L(\theta^{(i)})$$

→ ***The algorithm converges to local minima under mild assumptions*** ☺



- ***Spurious local minima*** cannot always be avoided
- Many variants have been derived to limit them, and to speed up convergence

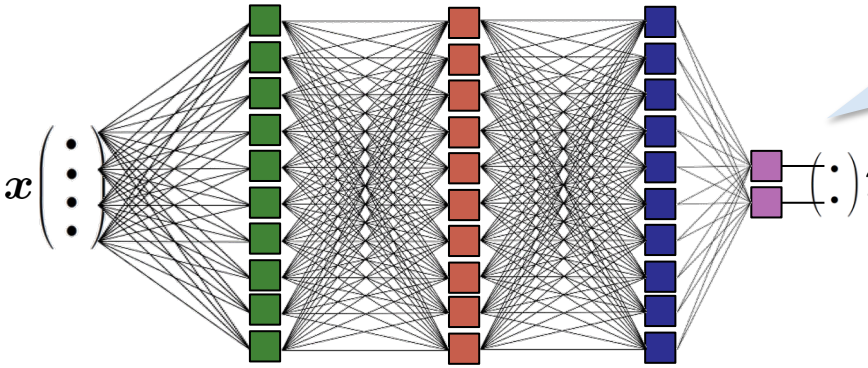
- The ***gradient-step*** ϵ, also called ***learning rate***, is a critical ***hyper-parameter***



Small ϵ | Large ϵ | Optimal ϵ

Image by Gabriel Peyre

Problem: How to efficiently compute the gradient of a DNN with respect to its parameters?



Compute $\nabla_{\theta} L(\theta, \mathcal{T})$ *?!* 😱

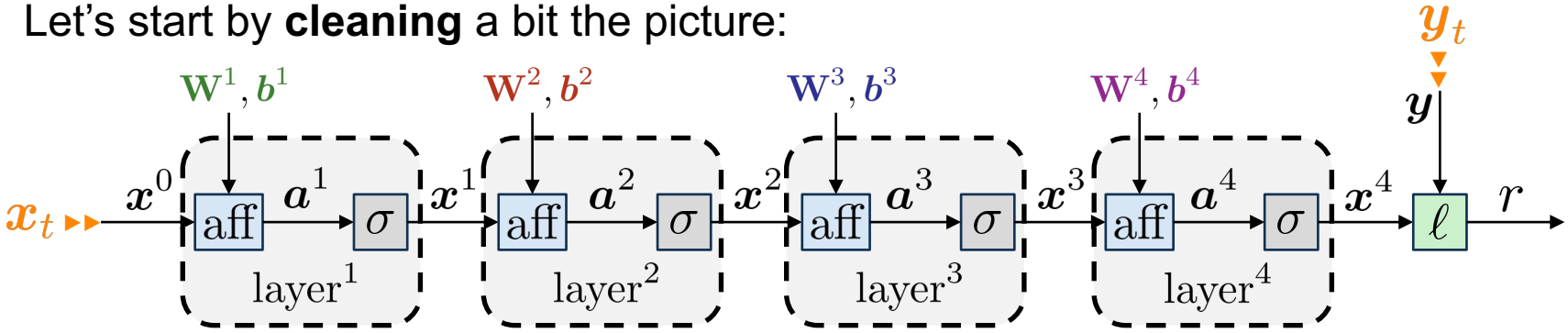
- Parameters:
$$\theta = [\theta^i]_{i=1}^4 = [(b^i, \mathbf{W}^i)]_{i=1}^4$$
- Training set:
$$\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$$
- Total Loss function:
$$L(\theta, \mathcal{T}) = \sum_{t=1}^{T} \ell(\mathrm{dnn}_{\theta}(x_t), y_t)$$

The *Backpropagation* Algorithm

- Uses the **chain rule** to compute the gradient layer by layer
- **Forward step**: send a data point (x_t, y_t) through the network's layers
- **Backward step:** compute the loss gradient at this point by recursively multiplying intermediate gradient values

The Backpropagation Algorithm

Let's start by **cleaning** a bit the picture:

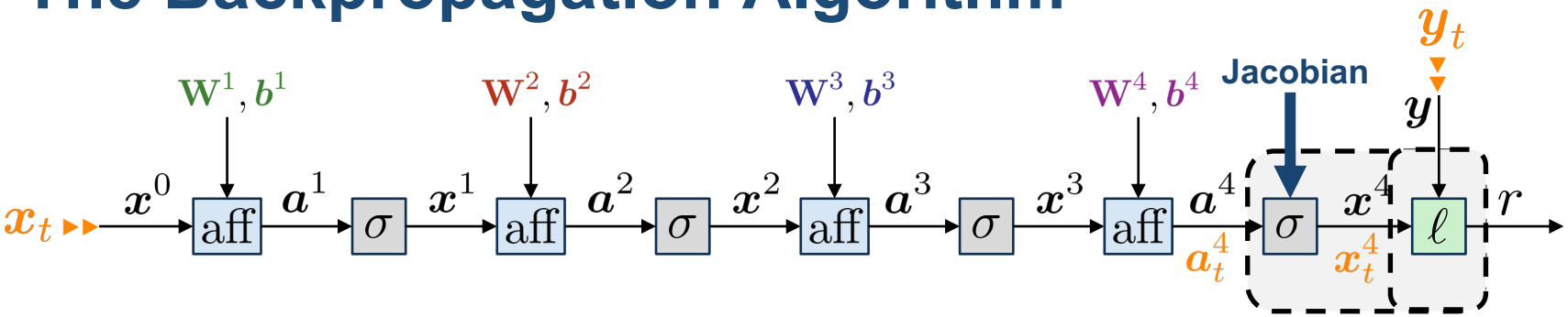


- $a^i = \mathrm{aff}_{\theta^i}(x^{i-1}) = \mathbf{W}^i x^{i-1} + b^i$ are the **pre-activations**
- $x^i = \sigma(a^i) = \sigma(\mathrm{aff}_{\theta^i}(x^{i-1})) = \mathrm{layer}^i(x^{i-1})$ are the **activations**
- The **loss** ℓ can be viewed as another layer, with **real output** r (the *"residual"*)
- Again, we have: $\nabla_\theta L(\theta, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \nabla_\theta \ell(\mathrm{dnn}_\theta(x_t), y_t)$
- It's enough to calculate the gradient of the loss at **one sample** (x_t, y_t):

$$G_\theta \overset{\text{def}}{=} \nabla_\theta \ell(\mathrm{dnn}_\theta(x_t), y_t)$$

and then sum.

The Backpropagation Algorithm



0) We start by $\boldsymbol{G}_{\boldsymbol{x}^4} = \dfrac{\partial r}{\partial \boldsymbol{x}^4}\Big|_{\boldsymbol{x}_t^4}^{\top} = \nabla_{\boldsymbol{x}^4}\ell(\boldsymbol{x}_t^4, \boldsymbol{y}_t)$.

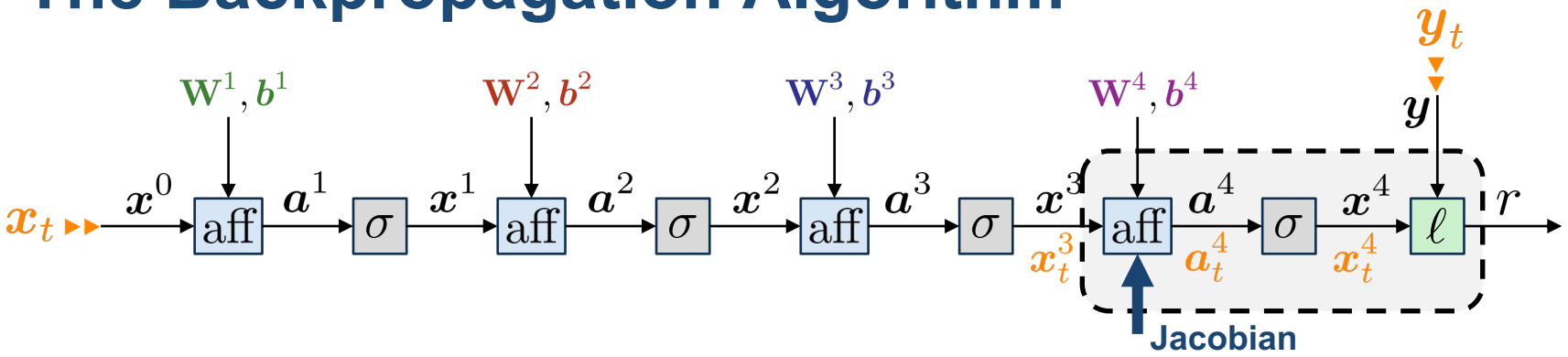
For example, for the L2 loss $\ell(\boldsymbol{x}^4, \boldsymbol{y}_t) = \|\boldsymbol{x}^4 - \boldsymbol{y}_t\|_2^2$, we have $\boldsymbol{G}_{\boldsymbol{x}^4} = 2(\boldsymbol{x}_t^4 - \boldsymbol{y}_t)$, the **difference** between the network **prediction** and the **target**.

1) Then, $\boldsymbol{G}_{\boldsymbol{a}^4} = \dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4}^{\top} = \left(\dfrac{\partial r}{\partial \boldsymbol{x}^4}\Big|_{\boldsymbol{x}_t^4} \times \dfrac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4}\right)^{\top} = \dfrac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4}^{\top} \times \boldsymbol{G}_{\boldsymbol{x}^4}$

$= \sigma'(\boldsymbol{a}_t^4) \odot \boldsymbol{G}_{\boldsymbol{x}^4}$.

$\dfrac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4}^{\top} = \mathrm{diag}[\sigma'(\boldsymbol{a}_t^4)]$!

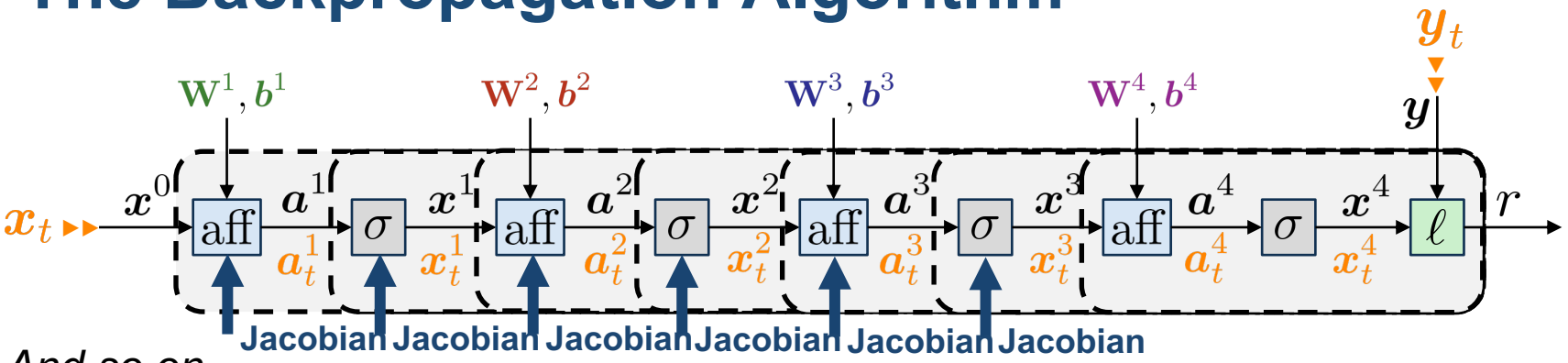
The Backpropagation Algorithm



2) Then:

- $\boldsymbol{G}_{\boldsymbol{x}^3} = \dfrac{\partial r}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}^\top = \left(\dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4} \times \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}\right)^\top = \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}^\top \times \boldsymbol{G}_{\boldsymbol{a}^4} = \mathbf{W}^{4\top}\boldsymbol{G}_{\boldsymbol{a}^4}$. $\quad \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}^\top = \mathbf{W}^4$
- $\boldsymbol{G}_{\boldsymbol{b}^4} = \dfrac{\partial r}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}^\top = \left(\dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4} \times \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}\right)^\top = \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}^\top \times \boldsymbol{G}_{\boldsymbol{a}^4} = \boldsymbol{G}_{\boldsymbol{a}^4}$. $\quad \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}^\top = \mathbf{I}$
- $\boldsymbol{G}_{\boldsymbol{w}_d^4} = \dfrac{\partial r}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}^\top = \left(\dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4} \times \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}\right)^\top = \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}^\top \times \boldsymbol{G}_{\boldsymbol{a}^4} = x_{t,d}^3 \boldsymbol{G}_{\boldsymbol{a}^4}$. $\quad \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}^\top = x_{t,d}^3 \mathbf{I}$

The Backpropagation Algorithm



And so on…

3) $\boldsymbol{G}_{\boldsymbol{a}^3} = \sigma'(\boldsymbol{a}_t^3) \odot \boldsymbol{G}_{\boldsymbol{x}^3}$.

4) $\boldsymbol{G}_{\boldsymbol{x}^2} = \mathbf{W}^{3\top} \boldsymbol{G}_{\boldsymbol{a}^3}$.

$\boldsymbol{G}_{\boldsymbol{b}^3} = \boldsymbol{G}_{\boldsymbol{a}^3}$.

$\boldsymbol{G}_{\boldsymbol{w}_d^3} = x_{t,d}^2 \boldsymbol{G}_{\boldsymbol{a}^3}$.

5) $\boldsymbol{G}_{\boldsymbol{a}^2} = \sigma'(\boldsymbol{a}_t^2) \odot \boldsymbol{G}_{\boldsymbol{x}^2}$.

6) $\boldsymbol{G}_{\boldsymbol{x}^1} = \mathbf{W}^{2\top} \boldsymbol{G}_{\boldsymbol{a}^2}$.

$\boldsymbol{G}_{\boldsymbol{b}^2} = \boldsymbol{G}_{\boldsymbol{a}^2}$.

$\boldsymbol{G}_{\boldsymbol{w}_d^2} = x_{t,d}^1 \boldsymbol{G}_{\boldsymbol{a}^2}$.

7) $\boldsymbol{G}_{\boldsymbol{a}^1} = \sigma'(\boldsymbol{a}_t^1) \odot \boldsymbol{G}_{\boldsymbol{x}^1}$.

8) $\boldsymbol{G}_{\boldsymbol{x}^0} = \mathbf{W}^{1\top} \boldsymbol{G}_{\boldsymbol{a}^1}$.

$\boldsymbol{G}_{\boldsymbol{b}^1} = \boldsymbol{G}_{\boldsymbol{a}^1}$.

$\boldsymbol{G}_{\boldsymbol{w}_d^1} = x_{t,d}^0 \boldsymbol{G}_{\boldsymbol{a}^1}$.

Putting everything together, we have for example:

$$\boldsymbol{G}_{\boldsymbol{b}^1} = \sigma'(\boldsymbol{a}_t^1) \odot \mathbf{W}^{2\top} \sigma'(\boldsymbol{a}_t^2) \odot \mathbf{W}^{3\top} \sigma'(\boldsymbol{a}_t^3) \odot \mathbf{W}^{4\top} \sigma'(\boldsymbol{a}_t^4) \odot \nabla_{\boldsymbol{x}^4} \ell(\boldsymbol{x}_t^4, \boldsymbol{y}_t) .$$

Stochastic Gradient Descent (SGD) and Batching

- We train DNNs using so called ***Empirical Risk Minimization***:

$$L(\boldsymbol{\theta}, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} \ell\left(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t\right) \approx \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}\{\ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{X}), \boldsymbol{Y})\}$$

- We compute the gradient of the **total loss** by summing the gradients of the **loss** at individual data samples:

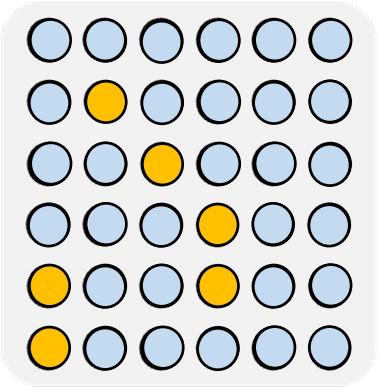
$$\nabla_{\boldsymbol{\theta}} L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$$

- But doing so across the **entire dataset** (e.g.: 1 million images) for **every gradient step** would be very expansive.

Stochastic Gradient Descent (SGD) and Batching

- At each iteration (i), compute the gradient over a **random subset** $\mathcal{T}^{(i)} \subseteq \mathcal{T}$ and perform one step of gradient descent:

$$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \cdot \frac{1}{|\mathcal{T}^{(i)}|} \sum_{(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}^{(i)}} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$$



- At next iteration, pick another (disjoint) random subset
- When the entire dataset has passed, start over again
- Each $\mathcal{T}^{(i)}$ is called a **minibatch**
- Each pass over the entire dataset is called an **epoch**

Splitting the training set into B minibatches:

- Reduces the computation cost of one gradient by a factor of B
- Increases the **standard deviation** on the gradient estimate by a factor of $\sqrt{B}$ only.

More iterations but fewer epochs = less total computation

+ compute gradients of 1 batch in parallel via GPU

The PyTorch framework

GOOD NEWS: You (probably) **won't ever need to implement backpropagation or SGD yourself!** ☺

- PyTorch: opensource Python library designed to easily **design**, **train** and **test** neural networks
- Initially developed by Facebook (Meta), based on Torch. **Constantly evolving** thanks to a broad community
- Seamlessly manipulate all the objects we have seen thanks to differential programming : **data, batches, parameters**, **layers, optimizers**, **loss…**
- Includes support for **GPU** and a C++ interface.
- Competing frameworks: **TensorFlow, Jax**

Outline

1. Introduction ✅

   AI? Machine Learning? Neural Networks?

2. Machine Learning ✅

   Definition, Characteristics, Types, Algorithms

3. Deep Neural Networks ✅

   Definition, Motivation, How to Train Them

4. Limits of Machine Learning

   Overfitting, Data Biases

Outline

1. Introduction

   AI? Machine Learning? Neural Networks?

2. Machine Learning

   Definition, Characteristics, Types, Algorithms

3. Deep Neural Networks

   Definition, Motivation, How to Train Them

4. Limits of Machine Learning

   Overfitting, Data Biases

Scenario:

- Imagine we have a training dataset $\mathcal{T}$ containing 10,000 images with labels (supervised learning)
- We train a machine learning model to perform the multi-class classification task
- We get nearly **perfect results** on these 10,000 images, e.g., 99.9% of correct classification
- However, when we run the model on new images, the results are **awful,** i.e., close to random.

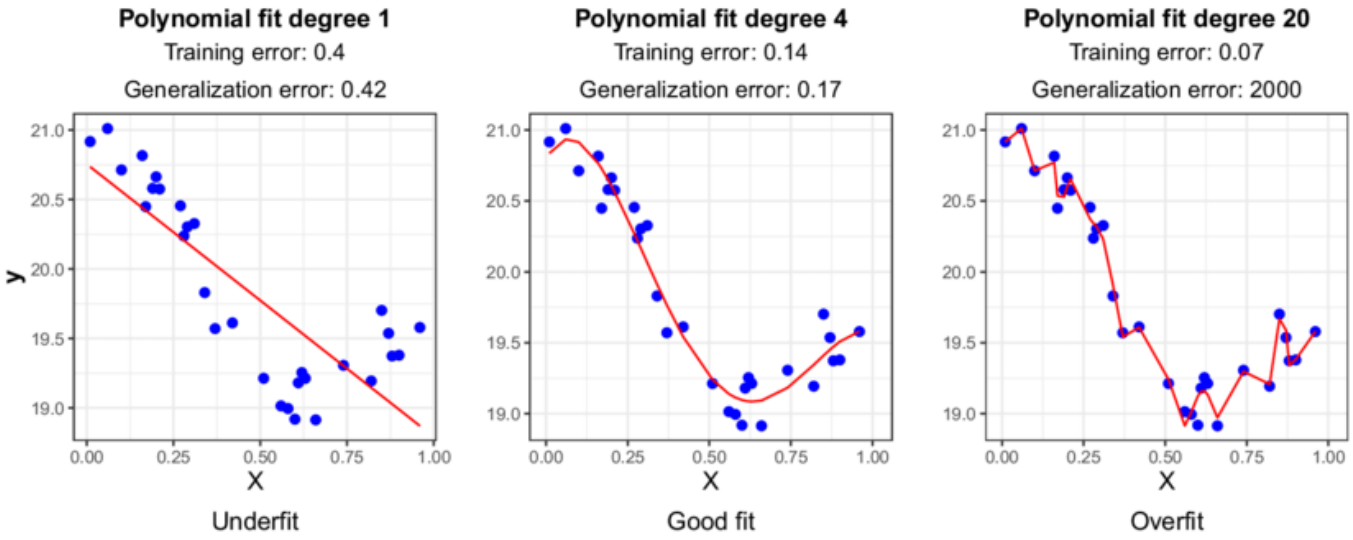
What's going on?

Overfitting

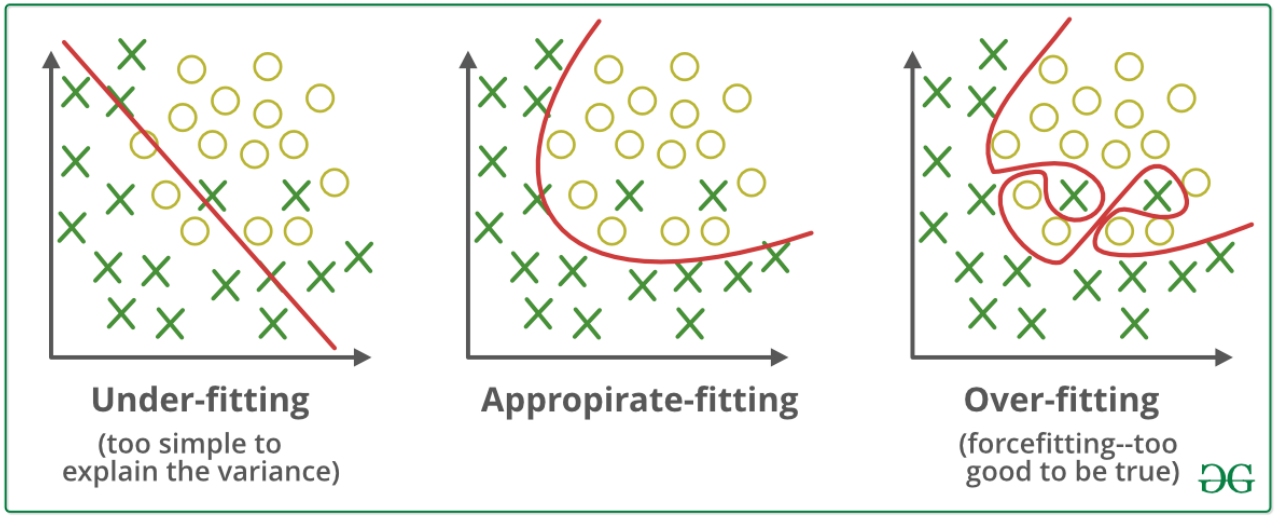
- Our algorithm is guilty of **overfitting**
- Instead of learning general features to classify the images, it **learned by heart** all the images in our training dataset!
- We sometimes have **millions** or **billions** of parameters in a model => it has the **capacity** to **store/encode** large amounts of data
- This may even happen for models of relatively small capacity, if the **amount of training data** is insufficient.

Overfitting

- Ex: polynomial regression

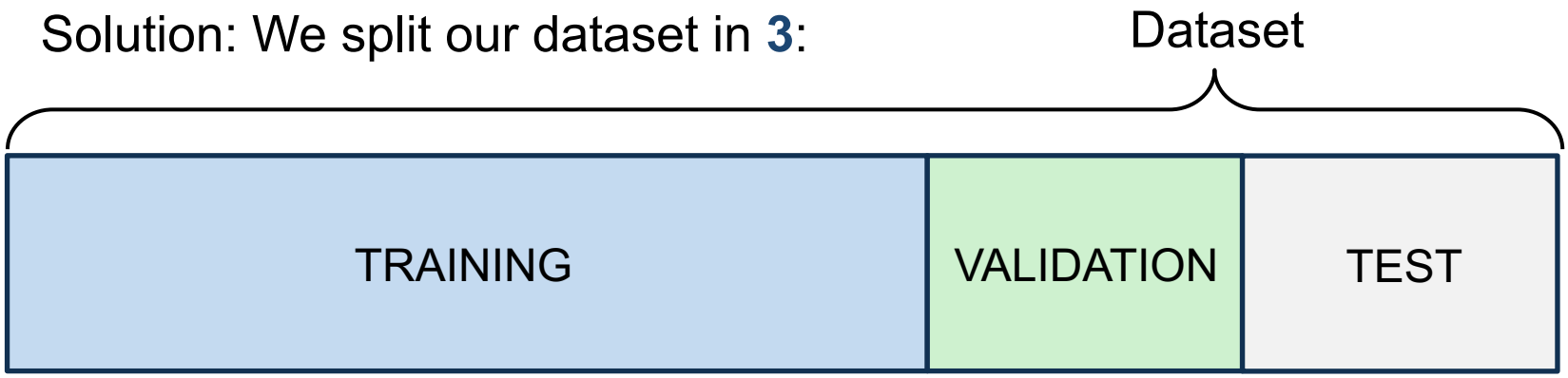


- Ex: binary classification



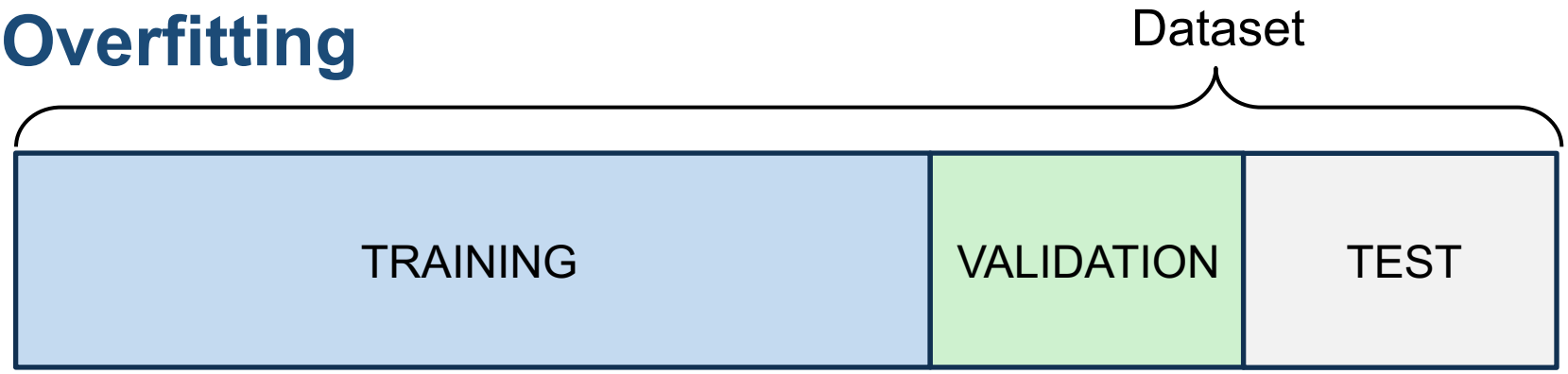
Overfitting

- In supervised learning, we are not really interested in a model that works perfectly on our training set. **We already have the answers anyway**, by definition of a (supervised) training set !
- We want a model that **generalizes** to **unseen** data
- Solution: We split our dataset in **3**:



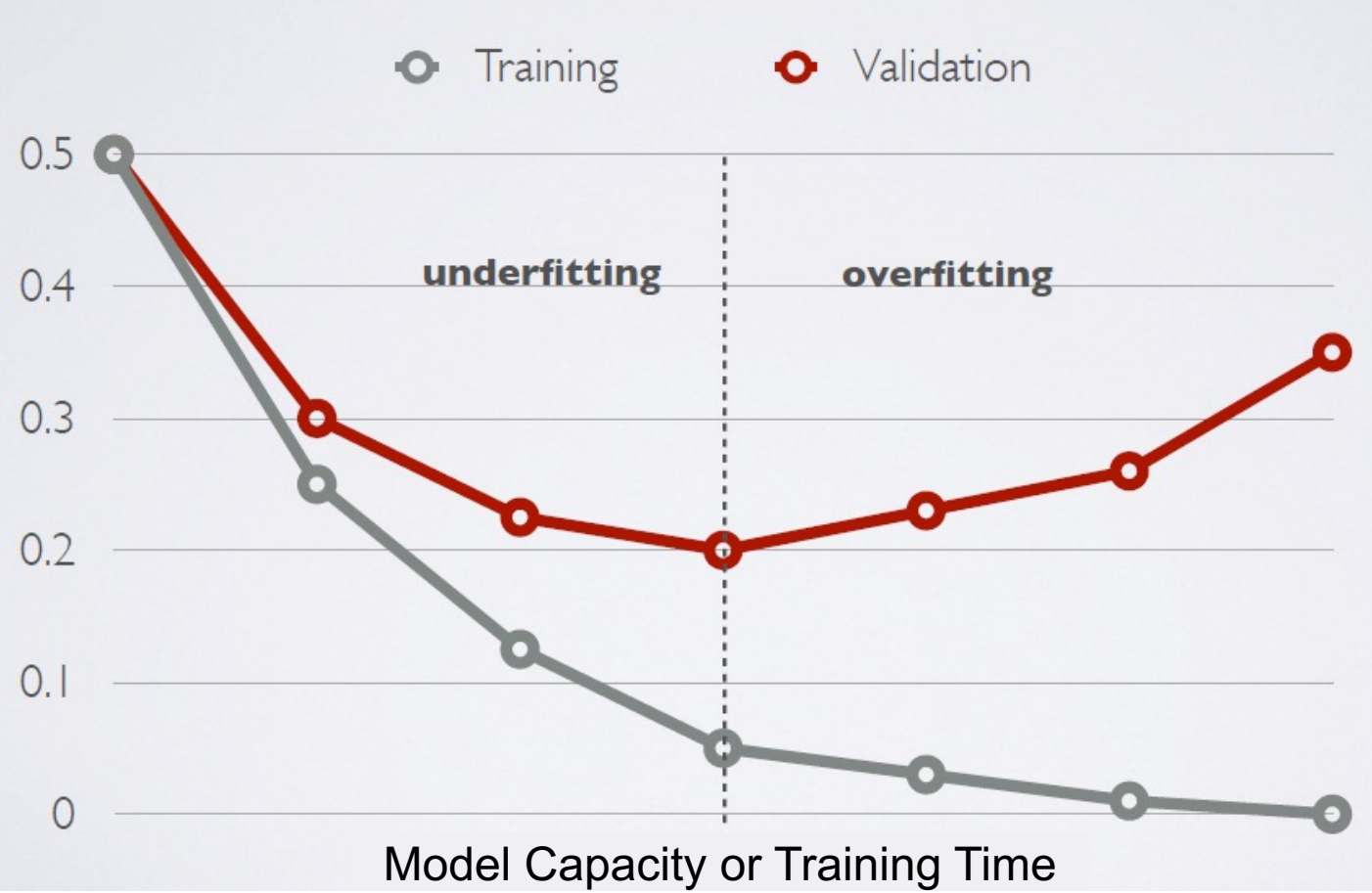
- These 3 subsets must be **perfectly disjoint** and all **representative** of the data.
- To achieve this, the split is done **at random**.

Overfitting



- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work
- The model parameters are only optimized over the **training set**
- We use the validation set to:
  - Verify that the model is making progress **while training it**
  - Tune ***hyperparameters***
  - Compare different families of models (***model selection***)
- Looking at the test set if **forbidden** in any of those steps

Overfitting vs. Underfitting



Example of technique to reduce overfitting

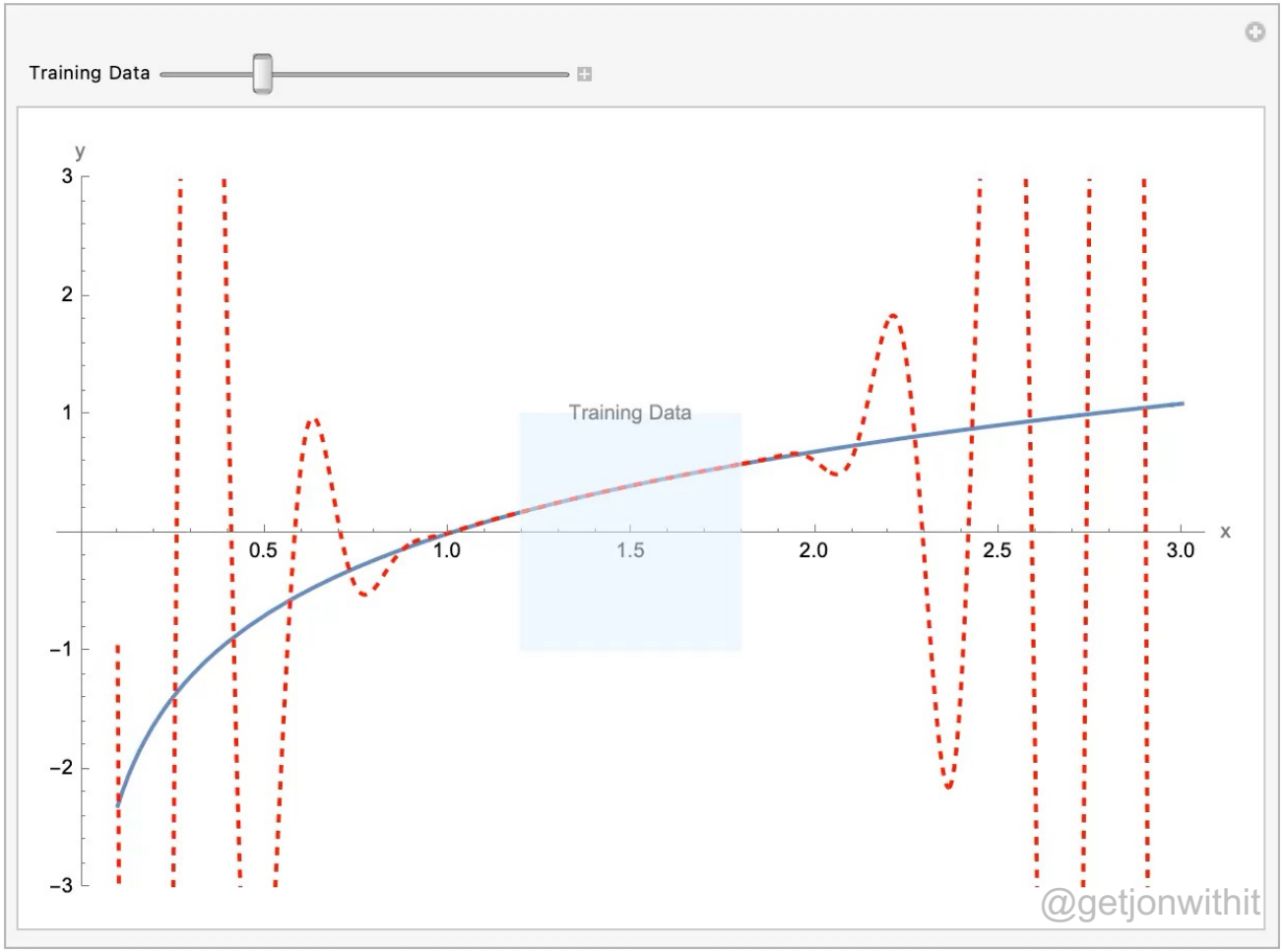
➔ Data augmentation

Signal transformations / Noise

Simulate data

Exercise: *Think of data augmentation strategies for your favorite acoustic application!*

- Machine learning models are **high-dimensional curve fitters**



- Don’t except them to **extrapolate** much beyond your training data

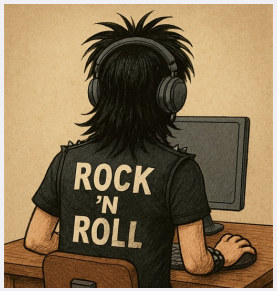
The « **Garbage in Garbage Out** » principle

➔ A machine learning model will only be as good as the data you give it

Beware of **data biases**

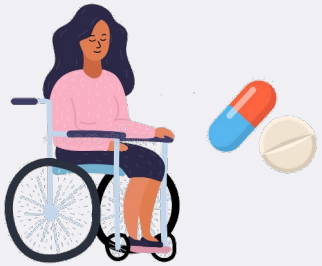
Representation bias

Measurement / Information bias



« Please label these songs according to their genre »

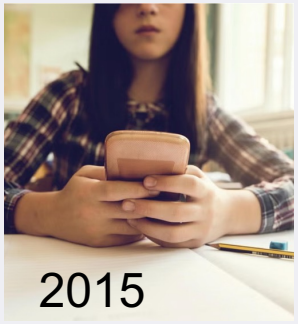
Exclusion bias



Prejudice bias

Automation bias

Historical bias



Impact of smartphone usage on school grades

- Machine learning aims at finding **models** from **data** to achieve **tasks** via **optimization** over **parameterized familes** of models
- Comes in **flavours**: supervised, unsupervised, predictive, generative, depending on the type of available data and task
- DNNs form a **versatile class of parameterized families of nonlinear functions** that can extract complex features from data
- **Inspired** (but far from matching!) biological brains
- DNNs are trained using variants of (stochastic) **gradient descent** and the **backpropagation algorithm**
- Made easy by modern deep learning framework such as **PyTorch**
- **Machine learning is only as good as the data you have !**

Now let's code !

ASSA AUTUMN SCHOOL SERIES IN ACOUSTICS